

7e

Systems Analysis AND Design

IN A CHANGING WORLD

JOHN SATZINGER | ROBERT JACKSON | STEPHEN BURD



SYSTEMS ANALYSIS AND DESIGN

In a Changing World

Seventh Edition

John W. Satzinger
Missouri State University

Robert B. Jackson
RBJ and Associates

Stephen D. Burd
University of New Mexico



Australia • Brazil • Mexico • Singapore • United Kingdom • United States

This is an electronic version of the print textbook. Due to electronic rights restrictions, some third party content may be suppressed. Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. The publisher reserves the right to remove content from this title at any time if subsequent rights restrictions require it. For valuable information on pricing, previous editions, changes to current editions, and alternate formats, please visit www.cengage.com/highered to search by ISBN#, author, title, or keyword for materials in your areas of interest.

Important Notice: Media content referenced within the product description or the product text may not be available in the eBook version.

Systems Analysis and Design in a Changing World, Seventh Edition**John W. Satzinger, Robert B. Jackson, Stephen D. Burd**

Vice President, General Manager, Social Science & Qualitative Business: Balraj Kalsi

Product Director: Joe Sabatino

Product Manager: Jason Guylar

Content Developer: Lori Bradshaw, S4Carlisle

Senior Product Assistant: Brad Sullender

Senior Market Manager: Eric La Scola

Marketing Coordinator: Will Guiliani

Art and Cover Direction, Production Management, and Composition:

Lumina Datamatics, Inc.

Intellectual Property Analyst: Christina Ciaramella

Senior Project Manager: Kathryn Kucharek

Manufacturing Planner: Ron Montgomery

Cover & Internal Image: Image Werks/Corbis

Uncredited figures are created by the authors.

© 2016, 2012 Cengage Learning

WCN: 02-200-203

ALL RIGHTS RESERVED. No part of this work covered by the copyright herein may be reproduced, transmitted, stored, or used in any form or by any means graphic, electronic, or mechanical, including but not limited to photocopying, recording, scanning, digitizing, taping, Web distribution, information networks, or information storage and retrieval systems, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the publisher.

For product information and technology assistance, contact us at
Cengage Learning Customer & Sales Support, 1-800-354-9706

For permission to use material from this text or product,
submit all requests online at **www.cengage.com/permissions**

Further permissions questions can be emailed to
permissionrequest@cengage.com

Library of Congress Control Number: 2014958278

ISBN: 978-1-305-11720-4

Cengage Learning

20 Channel Center Street

Boston, MA 02210

USA

Cengage Learning is a leading provider of customized learning solutions with office locations around the globe, including Singapore, the United Kingdom, Australia, Mexico, Brazil, and Japan. Locate your local office at: **www.cengage.com/global**

Cengage Learning products are represented in Canada by
Nelson Education, Ltd.

To learn more about Cengage Learning Solutions, visit **www.cengage.com**

Purchase any of our products at your local college store or at our
preferred online store **www.cengagebrain.com**

DEDICATION

To my wife JoAnn—JWS

To my immediate and extended family—RBJ

To Dee, Amelia, and Alex—SDB



BRIEF CONTENTS

PART ONE Introduction to System Development

- 1** From Beginning to End: An Overview of Systems Analysis and Design 3

Online Chapter A The Role of the Systems Analyst OL-1

PART TWO Systems Analysis Activities

- 2** Investigating System Requirements 37
- 3** Identifying User Stories and Use Cases 69
- 4** Domain Modeling 93
- 5** Use Case Modeling 131

Online Chapter B The Traditional Approach to Requirements OL-19

PART THREE Essentials of Systems Design

- 6** Foundations for Systems Design 157
- 7** Defining the System Architecture 185
- 8** Designing the User Interface 217
- 9** Designing the Database 257

PART FOUR System Development and Project Management

- 10** Approaches to System Development 295
- 11** Project Planning and Project Management 325

Online Chapter C Project Management Techniques OL-53

PART FIVE Advanced Design and Deployment Concepts

- 12** Object-Oriented Design: Fundamentals 365
- 13** Object-Oriented Design: Use Case Realization 397
- 14** Deploying the New System 443

Index 479



CONTENTS

Preface xviii

PART ONE Introduction to System Development

1 From Beginning to End: An Overview of Systems Analysis and Design 3

- Software Development and Systems Analysis and Design 4
- The System Development Life Cycle (SDLC) 7
- Iterative Development 8
- Introduction to Ridgeline Mountain Outfitters (RMO) 9
- Developing RMO's Tradeshow System 11
- Where You Are Headed—The Rest of This Book 28
- Chapter Summary 30
- Key Terms 30
- Review Questions 30
- Problem and Exercises 31
- Chapter case 31

Online Chapter A The Role of the Systems Analyst OL-1

- Overview OL-2
- The Analyst as a Business Problem Solver OL-3
- Systems That Solve Business Problems OL-6
- Required Skills of the Systems Analyst OL-10
- Analysis-Related Careers OL-13
- Chapter Summary OL-15
- Key Terms OL-16
- Review Questions OL-16
- Problem and Exercises OL-16
- Case Study OL-17

PART TWO Systems Analysis Activities

2 Investigating System Requirements 37

Overview	38
The RMO Consolidated Sales and Marketing System Project	39
Systems Analysis Activities	42
What Are Requirements?	45
Stakeholders	47
Information-Gathering Techniques	50
Models and Modeling	58
Documenting Workflows with Activity Diagrams	60
Chapter Summary	63
Key Terms	63
Review Questions	64
Problems and Exercises	64
Case Study	65
Running Case Studies	66
Further Resources	68

3 Identifying User Stories and Use Cases 69

Overview	70
User Stories and Use Cases	71
Use Cases and the User Goal Technique	73
Use Cases and Event Decomposition	74
Use Cases in the Ridgeline Mountain Outfitters Case	80
Chapter Summary	87
Key Terms	88
Review Questions	88
Problems and Exercises	88
Case Study	90
Running Case Studies	90
Further Resources	92

4 Domain Modeling 93

Overview	94
“Things” in the Problem Domain	94
The Entity-Relationship Diagram	100
The Domain Model Class Diagram	103
The State Machine Diagram—Identifying Object Behavior	114
Chapter Summary	122
Key Terms	123
Review Questions	123

Problems and Exercises	124
Case Study	126
Running Case Studies	127
Further Resources	129

5 Use Case Modeling 131

Overview	132
Use Case Descriptions	133
Activity Diagrams for Use Cases	137
The System Sequence Diagram—Identifying Inputs and Outputs	139
SSD Notation	140
Use Cases and CRUD	146
Integrating Requirements Models	148
Chapter Summary	149
Key Terms	149
Review Questions	149
Problems and Exercises	150
Case Study	151
Running Case Studies	151
Further Resources	154

Online Chapter B The Traditional Approach to Requirements OL-19

Overview	OL-20
Traditional and Object-Oriented Views of Activities and Use Cases	OL-21
Data Flow Diagrams	OL-21
Documentation of DFD Components	OL-38
Locations and Communication through Networks	OL-47
Chapter Summary	OL-50
Key Terms	OL-50
Review Questions	OL-50
Problems and Exercises	OL-51
Case Study	OL-51
Further Resources	OL-52

PART THREE Essentials of Systems Design

6 Foundations for Systems Design 157

Overview	159
What Is Systems Design?	159
Design Activities	163
System Controls and Security	168
Chapter Summary	179

- Key Terms 180
- Review Questions 180
- Problems and Exercises 180
- Case Study 181
- Running Case Studies 181
- Further Resources 183

7 Defining the System Architecture 185

- Overview 186
- Anatomy of a Modern System 187
- Architectural Concepts 195
- Interoperability 201
- Architectural Diagrams 201
- Describing the Environment 203
- Designing Application Components 208
- Chapter Summary 213
- Key Terms 213
- Review Questions 213
- Problems and Exercises 214
- Case Study 214
- Running Case Studies 215
- Further Resources 216

8 Designing the User Interface 217

- Overview 218
- Understanding the User Experience and the User Interface 219
- Fundamental Principles of User-Interface Design 223
- Transitioning from Analysis to User-Interface Design 232
- User-Interface Design 237
- Designing Reports, Statements, and Turnaround Documents 245
- Chapter Summary 251
- Key Terms 251
- Review Questions 251
- Problems and Exercises 252
- Case Study 253
- Running Case Studies 253
- Further Resources 255

9 Designing the Database 257

- Overview 258
- Databases and Database Management Systems 258

Database Design and Administration	260
Relational Databases	263
Distributed Database Architectures	279
Protecting the Database	284
Chapter Summary	286
Key Terms	287
Review Questions	287
Problems and Exercises	288
Case Study	289
Running Case Studies	290
Further Resources	292

PART FOUR System Development and Project Management

10 Approaches to System Development 295

Overview	296
The System Development Life Cycle	297
Methodologies, Models, Tools, and Techniques	301
Agile Development	304
The Unified Process, Extreme Programming, and Scrum	307
Chapter Summary	319
Key Terms	319
Review Questions	319
Problems and Exercises	320
Case Study	321
Running Case Studies	321
Further Resources	324

11 Project Planning and Project Management 325

Overview	326
Principles of Project Management	327
Activities of Core Process 1: Identify the Problem and Obtain Approval	335
Activities of Core Process 2: Plan and Monitor the Project	345
Chapter Summary	356
Key Terms	357
Review Questions	357
Problems and Exercises	357
Case Study	360
Running Case Studies	360
Further Resources	362

Online Chapter C Project Management Techniques OL-53

- Overview OL-54
- Calculating Net Present Value, Payback Period, and Return on Investment OL-55
- Understanding PERT/CPM Charts OL-58
- Building the Project Schedule with Microsoft Project OL-62
- Project Management Body of Knowledge (PMBOK) OL-70
- Chapter Summary OL-77
- Key Terms OL-77
- Review Questions OL-78
- Problems and Exercises OL-78
- Case Study OL-81

PART FIVE Advanced Design and Deployment Concepts

12 Object-Oriented Design: Fundamentals 365

- Overview 366
- Object-Oriented Design: Bridging from Analysis to Implementation 367
- Steps of Object-Oriented Design 374
- Design Classes and the Design Class Diagram 376
- Designing with CRC Cards 382
- Fundamental Principles for Good Design 388
- Chapter Summary 393
- Key Terms 393
- Review Questions 393
- Problems and Exercises 394
- Case Study 394
- Running Case Studies 395
- Further Resources 396

13 Object-Oriented Design: Use Case Realization 397

- Overview 398
- Object-Oriented Design with Interaction Diagrams 399
- Use Case Realization with Communication Diagrams 401
- Use Case Realization with Sequence Diagrams 408
- Developing a Multilayer Design 417
- Updating and Packaging the Design Classes 424
- Design Patterns 427
- Chapter Summary 434
- Key Terms 434
- Review Questions 434
- Problems and Exercises 435

Case Study	440
Running Case Studies	440
Further Resources	442

14 Deploying the New System 443

Overview	444
Testing	446
Deployment Activities	454
Managing Implementation, Testing, and Deployment	460
Putting It All Together—RMO Revisited	471
Chapter Summary	474
Key Terms	474
Review Questions	474
Problems and Exercises	475
Case Study	475
Running Case Studies	476
Further Resources	478

Index	479
-------	-----

FEATURES

Systems Analysis and Design in a Changing World, Seventh Edition, was written and developed with instructor and student needs in mind. Here is just a sample of the unique and exciting features that help bring the field of systems analysis and design to life.

BRIEF CONTENTS

PART ONE Introduction to System Development

1 From Beginning to End: An Overview of Systems Analysis and Design 3

Online Chapter A The Role of the Systems Analyst OL-1

PART TWO Systems Analysis Activities

2 Investigating System Requirements 37

3 Identifying User Stories and Use Cases 69

4 Domain Modeling 93

5 Use Case Modeling 131

Online Chapter B The Traditional Approach to Requirements OL-19

PART THREE Essentials of Systems Design

6 Foundations for Systems Design 157

7 Defining the System Architecture 185

8 Designing the User Interface 217

9 Designing the Database 257

PART FOUR System Development and Project Management

10 Approaches to System Development 295

11 Project Planning and Project Management 325

Online Chapter C Project Management Techniques OL-53

PART FIVE Advanced Design and Deployment Concepts

12 Object-Oriented Design: Fundamentals 365

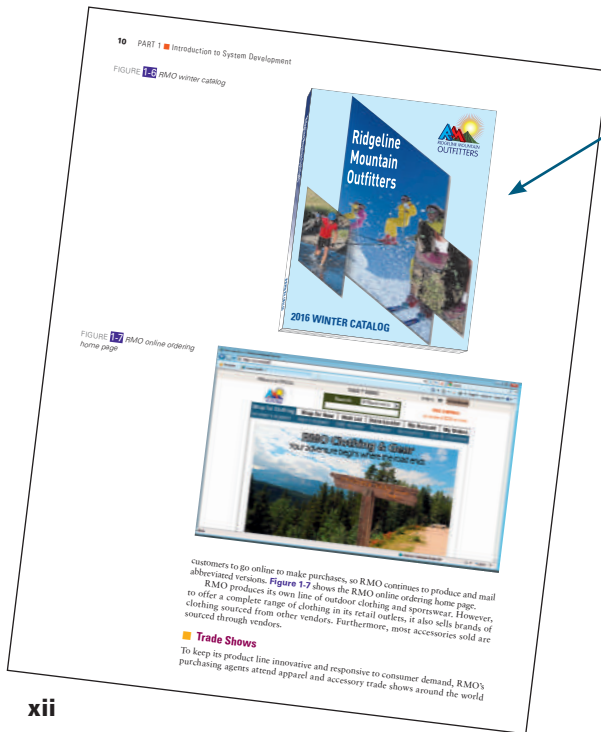
13 Object-Oriented Design: Use Case Realization 397

14 Deploying the New System 443

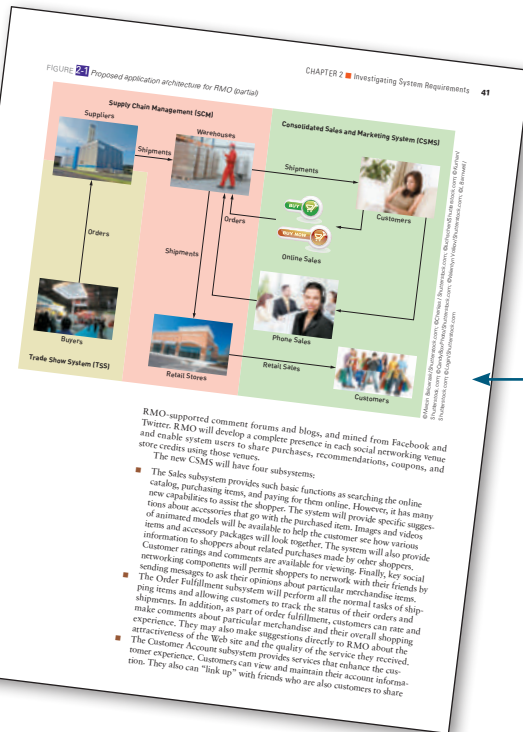
Index 479

The **innovative text organization** starts with a complete beginning-to-end system development example, moves immediately to systems analysis models and techniques, and then moves to systems design concepts emphasizing system architecture, user-interface design, and database design. Analysis and much of design is covered in the first nine chapters. Next, the text focuses on managing system development projects, including project planning and project management, after the student has a chance to learn what is involved in system development. Finally, the text covers detailed object-oriented design techniques and deployment topics.

The text uses a **completely updated** integrated case study of moderate complexity—**Ridgeline Mountain Outfitters (RMO)**—to illustrate key concepts and techniques. In addition, a smaller RMO application—the Tradeshow System—is used in Chapter 1 to introduce the entire system development process.

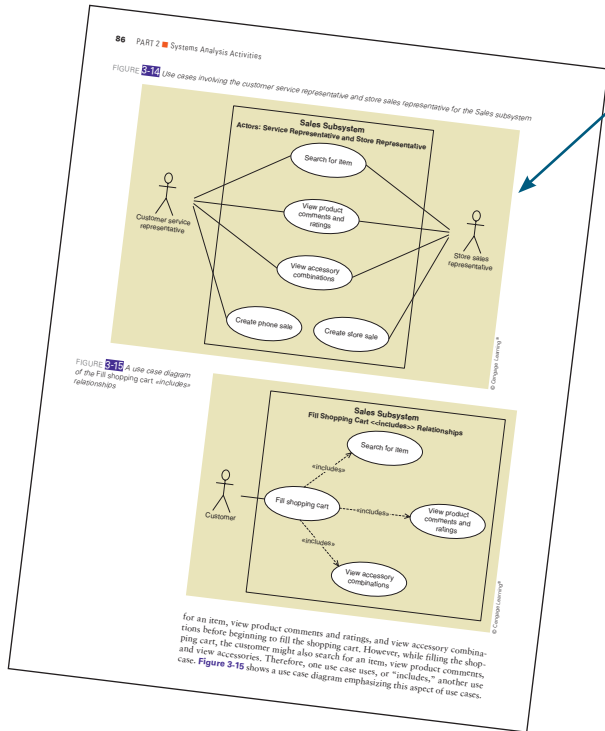


FEATURES



The planned **RMO application architecture** provides for rich examples—a Web-based component, a wireless smartphone/tablet application, and a client/server Windows-based component. All RMO applications described are integrated and strategically planned. The **Supply Chain Management System** already exists, ready for integrating the **Tradeshow System** and the new **Consolidated Sales and Marketing System**.

The new **Consolidated Sales and Marketing System (CSMS)** is the system development project described in Chapter 2 and used throughout the text for examples and explanations. It is strategically important to RMO, and the company must integrate the new system with legacy systems and other planned systems. There are **four subsystems**, and the requirements and design models are shown in detail. UML diagrams are used throughout for examples and exercises.



FEATURES

The text describes both **predictive and adaptive approaches to the SDLC** and recommends **Agile, iterative development** for most projects. The SDLC used in the text features a generic, condensed version of the Unified Process SDLC taught as an Agile approach that emphasizes iterations and core development processes. Core development processes and iterations are emphasized over phases to reduce the confusion that ordinarily occurs when students are taught “phases” and then told to use iterations. **Project planning and project management** are emphasized throughout, and the book focuses more on systems analysis and systems design as development disciplines rather than phases.

Design Activities

Figure 6-3 identifies the activities of systems design. This section provides a short introduction to each of these design activities. In-depth explanation and instruction on the specific concepts and skills for each design activity are given later in the text.

Systems design involves specifying in detail how a system will work when deployed within a specific technology environment. Some of the details may have been defined during systems analysis, but much more detail is added during design. In addition, each part of the final solution is heavily influenced by the design of all the other parts. Thus, systems design activities are usually done in parallel. For example, the database design influences the design of application components, software classes and methods, and the user interface. Likewise, the technology environment drives many of the decisions for how system functions are distributed across application components and how those components communicate across a network. When an iterative approach to the SDLC is used, major design decisions are made in the first or second iteration; however, many decisions are revisited and refined during later iterations.

To better understand these design activities, you can summarize each one with a question. In fact, system developers often ask themselves these questions presents these questions:

FIGURE 6-3 Design activities

Design activities

- Describe the environment.
- Design the application components.
- Design user interface.
- Design the database.
- Design the software classes and methods.

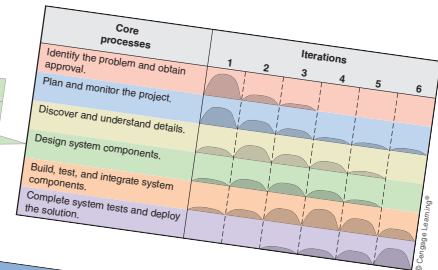


FIGURE 6-4 Design activities and key questions

Design activity	Key question
Describe the environment	How will this system interact with other systems and with the organization's existing technologies?
Design the application components	What are the key parts of the information system and how will they interact when the system is deployed?
Design the user interface	How will users interact with the information system?
Design the database	How will data be captured, structured, and stored for later use by the information system?
Design the software classes and methods	What internal structure for each application component will ensure efficient construction, rapid deployment, and reliable operation?

FIGURE 10-4 Overlap of system development phases

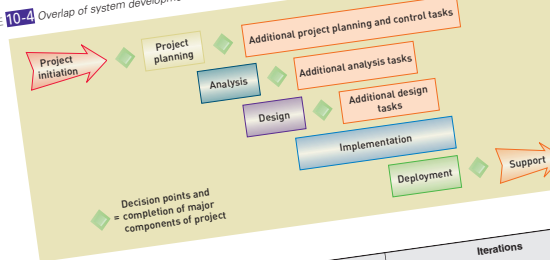
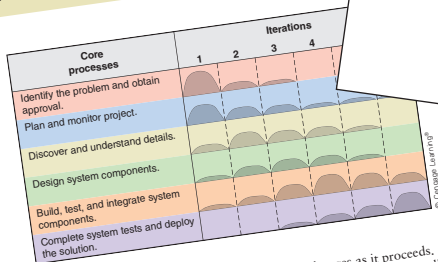


FIGURE 10-5 Adaptive SDLC with six core processes and multiple iterations



Using iterations, the project is able to adapt to any changes as it proceeds. Also, parts of the system are available early on for user evaluation and feedback, which helps ensure that the application will meet the needs of the users.

You first saw this concept in the SDLC example used in Chapter 1, which is repeated here as Figure 10-5. The core processes defined in Chapter 1 are carried out in each iteration of the project. This iterative approach, modification is carried out in each iteration's analysis, design, and implementation, variations can be made to adapt to the changing requirements of the project. The adaptive approach presented in this textbook is a simplification of and variations on a more formal iterative approach called the Unified Process (UP). You will learn more about the UP later in this chapter.

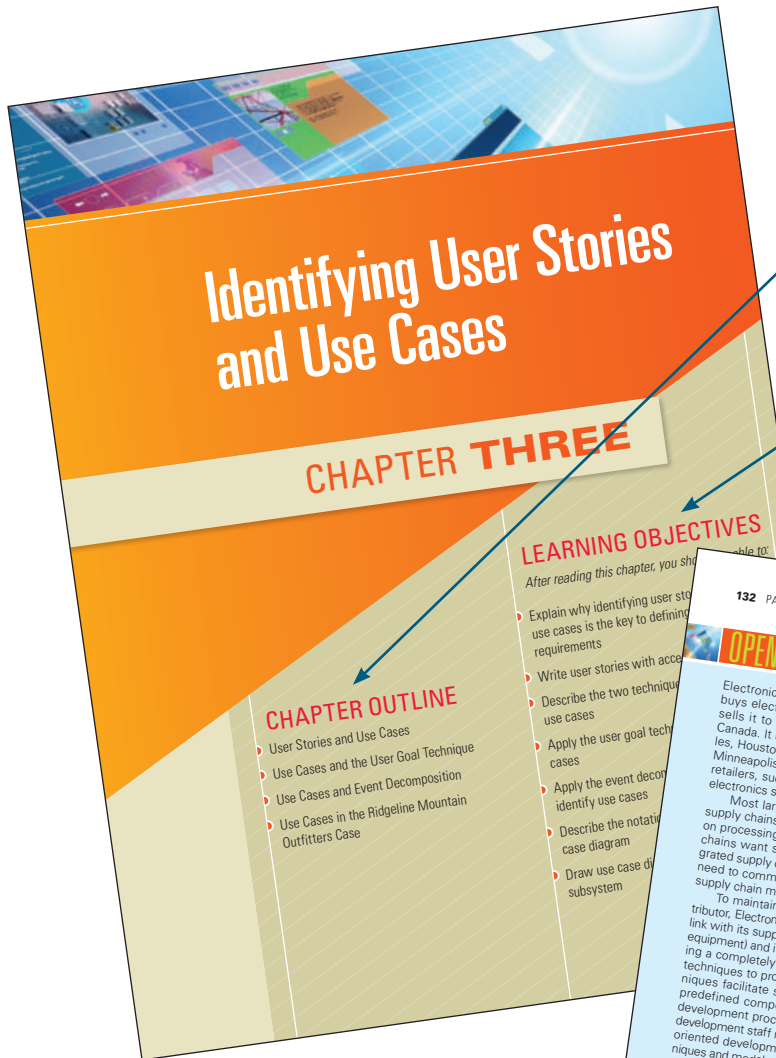
A related concept to an iterative SDLC is called **incremental development**. Incremental development is always based on an iterative life cycle. The basic idea is that the system is built in small increments. An increment may be developed within a single iteration or it may require two or three iterations. As each increment is completed, it is integrated with the whole. The system, in effect, is “grown” in an organic fashion. The advantage of this approach is that portions of the system get into the users’ hands much sooner so the business can begin accruing benefits as early as possible.

Yet another related concept, which is also based on an iterative approach, is the idea of a **walking skeleton**. A walking skeleton, as the name suggests, provides a complete front-to-back implementation of the new system but with only

incremental development an SDLC approach that completes portions of the system in small increments across iterations, with each increment being integrated into the whole as it is completed

walking skeleton a development approach in which the complete system structure is built but with bare-bones functionality

FEATURES



Each chapter provides a **chapter outline**, states clear **learning objectives**, and includes an **opening case study**.

CHAPTER OUTLINE

- User Stories and Use Cases
- Use Cases and the User Goal Technique
- Use Cases and Event Decomposition
- Use Cases in the Ridgeline Mountain Outfitters Case

LEARNING OBJECTIVES

- After reading this chapter, you should be able to:
- Explain why identifying user stories and use cases is the key to defining requirements
 - Write user stories with acceptance criteria
 - Describe the two techniques for identifying use cases
 - Apply the user goal technique to identify use cases
 - Apply the event decomposition technique to identify use cases
 - Describe the notation for use case diagrams
 - Draw use case diagrams for a subsystem

132 PART 2 ■ Systems Analysis Activities

OPENING CASE ELECTRONICS UNLIMITED: INTEGRATING THE SUPPLY CHAIN

Electronics Unlimited is a warehousing distributor that sells IT to retailers throughout the United States and Canada. It has operations and warehouses in Los Angeles, Houston, Baltimore, Atlanta, New York, Denver, and Minneapolis. Its customers range from large nationwide electronics stores, such as Target, to medium-sized independent supply chains. Information systems used to be focused on processing internal data; however, today, these retailers want suppliers to become part of a totally integrated supply chain system. In other words, the systems need to communicate between companies to make the supply chain more efficient.

To maintain its position as a leading wholesale distributor, Electronics Unlimited has to convert its system to link with its suppliers (the manufacturers of the electronic equipment) and its customers (the retailers). It is developing a completely new system that uses object-oriented techniques to provide these links. Object-oriented techniques facilitate system-to-system interfaces by using predefined components and objects to accelerate the development process. Fortunately, many of the system development staff members have experience with object-oriented development and are eager to apply the techniques and models to the system development project.

William Jones is explaining object-oriented development to the group of systems analysts who are being trained in this approach.

"We're developing most of our new systems by using object-oriented principles," he tells them. "The complexity of the new system, along with its interactivity, makes the object-oriented approach a natural way to develop requirements. The object-oriented models track

very closely with the new object-oriented programming languages and frameworks."

William is just getting warmed up. "This way of thinking about a system in terms of objects is very interesting," he adds. "It is also considerably first learned to think about objects when you developed screens for the user interface. All the controls on the screen, such as buttons, text boxes, and drop-down boxes, are objects. Each has its own set of trigger events that activate its program functions."

"How does this apply to our situation?" one of the analysts asks.

"You just extend that thought process," William explains. "You think of such things as purchase orders and employees as objects, too. We can call them the interface objects, such as windows and buttons. During analysis, we have to find out all the trigger events and methods associated with each business object."

"And how do we do that?" another analyst asks.

"You continue with your fact-finding activities and build a better understanding of each use case," William says. "The way the problem domain objects interact with each other in the use case determines how you identify the initiating activity. We refer to those activities as the messages between objects. The tricky part is that you need to think in terms of objects instead of just processes. Sometimes, it helps me to pretend I am an object. I will say, 'I am a purchase order object. What functions and services are other objects going to ask me to do?' After you get the hang of it, it works very well and it is enlightening to see how the system requirements unfold as you develop the diagrams."

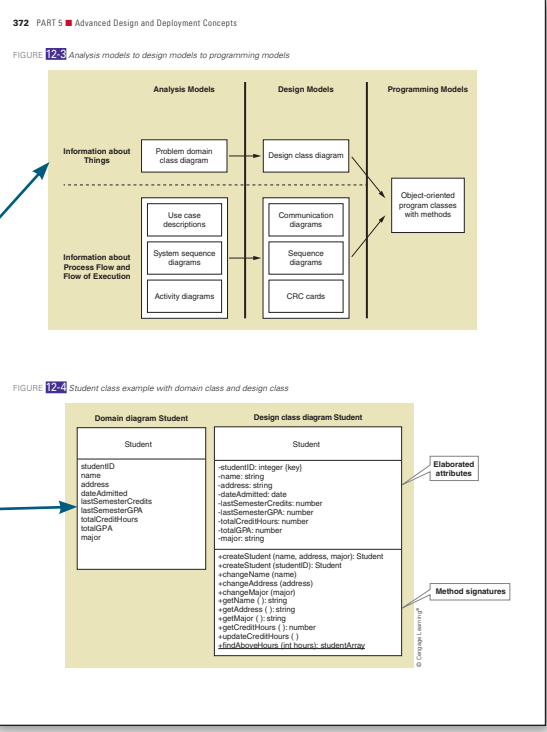
Overview

The main objective of defining requirements in system development is understanding users' needs, how the business processes are carried out, and how the system will be used to support those business processes. As indicated in Chapter 2, system developers use a set of models to discover and understand the requirements for a new system. This activity is a key part of systems analysis in the system development process. The first step in the process for developing this finding activities requires the fact-finding skills you learned in Chapter 2. Fact-finding activities are also called *discovery activities*, and obviously, discovery must precede understanding.

The models introduced in Chapters 3 and 4 focus on two primary aspects of functional requirements: the use cases and the problem domain classes involved in users' work. User stories are sometimes used in place of use cases with Agile development. Use cases are identified by using the user goal technique and the

FEATURES

Margin definitions of key terms are placed in the text when a term is first used. Each chapter includes extensive figures and illustrations designed to clarify and summarize key points and to provide examples of UML diagrams and other deliverables produced by an analyst.



72 PART 2 ■ System Analysis Activities

acceptance criteria features that must be present in the final system for the user to be satisfied

A final part of a user story is the **acceptance criteria**. These indicate the features that must be present for the user to be satisfied with the resulting implementation. They focus on functionality, not on features or user-interface design. For example, the following are the acceptance criteria for the user story "bank teller making a deposit".

1. Customer lookup must be by name or by account number.
2. It would be nice to display photo and signature of customer.
3. Any check hold requirements must be indicated.
4. Current balance and new balance must be displayed.

The programmer analyst uses the acceptance criteria to clarify the expectations of the user and to verify the user is looking at the user story at an appropriate level of analysis. When the user story is implemented and refined, the acceptance criteria are used for testing. Some consider it much like a contract between the developers and the users that limits controversy later in the project. Figure 3-4 shows two user stories handwritten on index cards. The first user story is for the bank teller example just discussed. The other user story is for a shipping clerk responsible for shipping the items on a new order for RMO.

FIGURE 3-4 Two user stories with acceptance criteria

User Story

As a *bank teller*, I want to *make a deposit to quickly serve more customers*.

Acceptance Criteria:

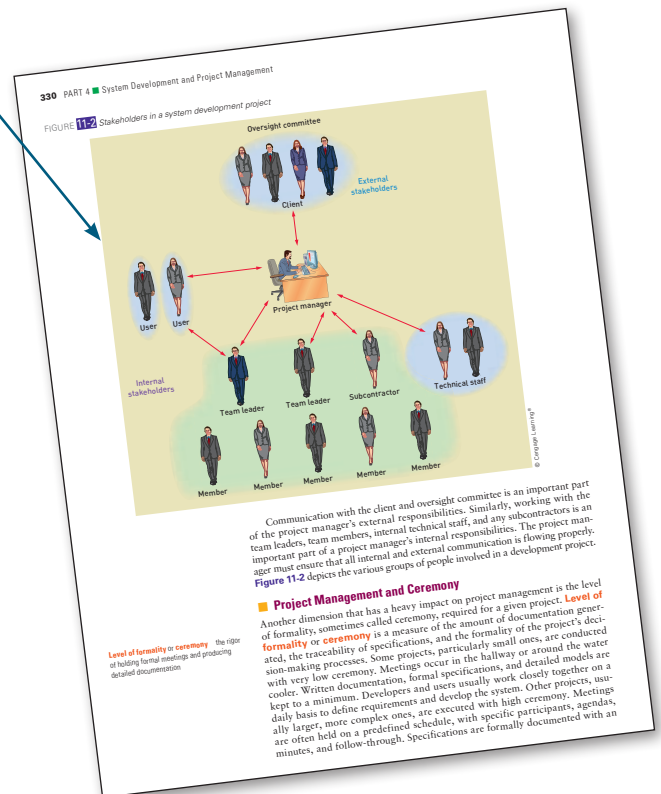
1. Customer lookup must be by name or by account number.
2. Nice to display photo and signature of customer.
3. Any check hold requirements must be indicated.
4. Current balance and new balance must be displayed.

User Story

As a *shipping clerk*, I want to *ship an order as accurately as possible as soon as the order details are available*.

Acceptance Criteria:

1. Available order details must pop up on the screen when available.
2. Portable display and scan device would cut time in half.
3. Sort the items by bin location.
4. Indicate number of items in stock for each item and mark backorder for those not available.
5. Recommend shipper based on weight, size, and location.
6. Print out shipping label for selected shipper.



FEATURES

End-of-chapter material includes a detailed summary, an indexed list of key terms, and ample review questions.

Each chapter also includes a collection of problems and exercises that involve additional research or problem solving, an end-of-chapter case study, four running cases that create challenging and integrated course assignments, and a list of further resources.

CHAPTER SUMMARY

CHAPTER 12 ■ Object-Oriented Design: Fundamentals 393

The ultimate responsibility of system developers is to write computer software that solves a business problem. This chapter focuses on how to configure and develop the solution system—that is, how to design the details of the new system. Systems design is the bridge that puts business requirements in terms that the programmers can use to write the software that becomes the solution system.

Using all the requirements models as well as the architectural design, object-oriented design extends the models so programming can proceed. The objective of object-oriented design is to determine the methods within individual classes that are needed to implement the use cases. The process of design is use-case-driven, in that it is done one use case at a time. The process of object-oriented design can be divided into two major areas: developing a design class use case via an interaction diagram. The DCD is usually developed in two steps. A first-cut DCD is created based on the domain model class diagram, but then it

is refined and expanded as the sequence diagrams are developed. One method of determining which objects collaborate is to use class responsibility collaboration (CRC) cards. For simple use cases, a set of CRC cards may be sufficient to write code. For more complex use cases, other interaction diagrams are normally used.

One reason that we suggest a more formal system of design, rather than just starting to write code is that the final system is much more robust and maintainable. Design as a rigorous activity builds better ideas as coupling and cohesion, and each of the classes has high cohesion. Another important principle is “protection from variations,” meaning that some parts of the system should be protected from and not too stable and subject to change. Being a good developer entails learning and following the principles of good design.

KEY TERMS

- boundary or view class
- class-level attribute
- class-level method
- cohesion
- controller class
- coupling
- CRC (class responsibility collaboration) cards
- data access class
- entity class
- indirection
- instantiation
- method signature
- navigation visibility
- object-oriented design
- object responsibility
- persistent class
- protection from variations
- separation of responsibilities
- stereotype
- visibility

REVIEW QUESTIONS

1. Describe in your own words how an object-oriented program works.
2. What is instantiation?
3. List the models that are used for object-oriented systems design.
4. Explain how domain classes are different from design classes.
5. What is the difference between a system sequence diagram and a sequence diagram?
6. In your own words, list the low steps for doing object-oriented design.
7. What do we mean by use-case-driven design?
8. Explain in your own words what coupling means and why it is important.
9. Explain what cohesion means and why it is important.
10. Compare and contrast the ideas of coupling and cohesion.
11. What is protection from variations, and why is it important in detailed design?
12. What is meant by object responsibility, and why is it important in detailed design?

FURTHER RESOURCES

- Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language Handbook*, Addison-Wesley, 1999.
- Grady Booch, James Rumbaugh, and Ivar Jacobson, *Object-Oriented Analysis and Design: A Practical Approach*, Addison-Wesley, 1999.
- E. Reed Doherty, *UML 2.0 Superstructure: Using Object-Oriented Application Development*, Addison-Wesley, 2002.
- Philippe Kruchten, *The Rational Unified Process: An Introduction* (3rd ed.), Addison-Wesley, 2005.
- Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process* (3rd ed.), Prentice Hall, 2005.
- Object Management Group, *UML 2.0 Superstructure Specification*, 2004.

150 PART 2 ■ Systems Analysis Activities

14. What are two ways to show repetition on a sequence diagram?
15. What are the three types of frames used on a sequence diagram?
16. What is the symbol for a true/false condition on a sequence diagram?
17. Explain what parameters of a message are.

PROBLEMS AND EXERCISES

1. After reading the following narrative, do the following:

- a. Develop an activity diagram for each scenario.
- b. Complete a fully developed use case description for each scenario.

Quality Building Supply has two kinds of customers: contractors and the general public. A contractor buys materials by taking them to the checkout desk for contractors. The clerk then opens up a new ticket for the contractor. Next, the clerk scans (in each item to be purchased) the item to the ticket. At the end of the sale, the clerk indicates the end of the sale, the contractor's current credit limit and, if it is acceptable, finalizes the sale. The system creates the sale, some contractors like to keep a record of their purchases, so they request that ticket details be printed. Others aren't interested in a printout.

A sale to the general public is simply entered into the cash register, and a paper ticket must be made by cash, check, or credit card. The clerk must enter the type of payment to ensure that the cash register balances at the end of the shift. For credit card payments, the system prints a credit card voucher that the customer must sign.

2. Based on the following narrative, develop either an activity diagram or a fully developed description for the use case *Add a new vehicle to an existing policy* in a car insurance system.

A customer calls a clerk at the insurance company and gives his policy number. The clerk enters this information, and the system displays the basic insurance policy. The clerk then checks

18. List the primary steps for developing an SSD.
19. What are the words included in the CRUD acronym?
20. What is the purpose of using the CRUD technique?
21. Identify the models explained in this chapter and their relationship to one another.

4. Develop an SSD based on the narrative and your activity diagram for problem 1.

The development of an SSD based on the narrative or your activity diagram for problem 2.

Develop a policy in your area that develops with large staff of information systems professionals. The system must be more rigorous in their development. Set up an interview. Determine the development process. Use traditional structured analysis. Use UML 2 Toolkit. John Wiley & Sons, 2004.

CASE STUDY

TheEyesHavelt.com Book Exchange

TheEyesHavelt.com Book Exchange is a type of e-business that does business entirely on the Internet. The company acts as a clearinghouse for buyers and sellers of used books.

When a person registers with theEyesHavelt.com, the person must provide a current physical address and telephone number as well as a current e-mail address. The system then maintains an open account for this person. Access to the system as a seller is through a secure, authenticated portal.

TheEyesHavelt.com lists books on the system through a special form. The form asks for all the pertinent information about the book: its category, its general condition, and the asking price. A seller may list as many books as desired. The system maintains an index of all books in the system so buyers can use the search engine to search for books. The search engine allows searches by title, author, ISBN, and keyword.

Anyone who wants to buy books come to the site and search for the books they want. When they decide to buy, they must open an account with a credit card to pay for the books. The system maintains all this information on secure servers.

When a purchase is made, TheEyesHavelt.com sends an e-mail notice to the seller of the book that was chosen as well as payment information. It also marks the book as sold. The system maintains an open order until it receives notice that the book has been shipped. After the seller receives notice that a listed book has been sold, the seller must notify the buyer via e-mail within 48 hours that the purchase is noted. Shipment of the order must be made within 24 hours after the seller sends the notification e-mail. The seller sends a notification to the buyer and TheEyesHavelt.com when the shipment is made.

After receiving the notice of shipment, TheEyesHavelt.com maintains the order in a shipped status. At the end of each month, a check is mailed to each seller for the book orders that have remained in a shipped status for 30 days. The 30-day waiting period exists to allow the buyer to notify TheEyesHavelt.com if the shipment doesn't arrive for some reason or if the book isn't in the same condition as advertised.

If they want, buyers can enter a service code for the seller. The service code is an indication of how well the seller is servicing book purchases. Some sellers are very active and use TheEyesHavelt.com as a major outlet for selling books. Thus, a service code is an important indicator to potential buyers.

For this case, develop these diagrams:

1. A domain model class diagram
2. A list of use cases and a use case diagram
3. A fully developed description for two use cases: *Add a seller and Record a book order*
4. An SSD for each of the two use cases: *Add a seller and Record a book order*

RUNNING CASE STUDIES

Community Board of Realtors®

The Multiple Listing Service system has a number of use cases, which you identified in Chapter 3, and three key domain classes, which you identified in Chapter 4: *RealEstateOffice*, *Agent*, and *Listing*.

1. For the use case *Add agent to real estate office*, write a fully developed use case description. Also develop an activity diagram and draw an SSD. Review the case materials in previous



PREFACE

When we wrote the first edition of this textbook, the world of system development was in a major transition period—from structured methodologies to object-oriented methodologies. We were among the first to introduce a comprehensive treatment of object-oriented methodologies, and *Systems Analysis and Design in a Changing World, Seventh Edition*, continues to be the leader in teaching UML and object-oriented techniques.

However, change continues. Today, many new initiatives and trends have become firmly embedded in the world of system development. First and foremost is the ubiquitous access to the Internet throughout the global economy. The resulting explosion of connectivity means that project teams are now distributed around the world. In addition, large providers (such as Microsoft) and a proliferation of small providers now contribute to a wonderfully rich and varied software development environment.

In order to manage system development teams in today's distributed, fast-paced, connected, ever-changing environment, the techniques for software development and the approach to project management have expanded. Along with the foundational project management principles, additional approaches and philosophies provide new, success-oriented methodologies, such as Agile iterative, incremental development approaches. These are thoroughly covered in this edition.

Even though *Systems Analysis and Design in a Changing World, Seventh Edition*, continues to be the leader in its field, with thorough treatment of such topics as user stories, use cases, object-oriented modeling, comprehensive project management, the Unified Modeling Language, and Agile techniques, it was time to take another step forward in textbook design. This edition uses an innovative approach to teaching systems analysis and design, taking advantage of the new teaching tools and techniques that are now available. As a result, not only is systems analysis and design easier to learn by using this approach, it is also easier to teach. It brings together the best approaches for teachers *and* students.

In this edition, we accomplish four major new objectives. First, we teach all the essential principles of system development—principles that must be followed in today's connected environment. Second, we teach and explain the new methodologies and techniques that are now available because of widespread connectivity. Third, we have organized and revamped the textbook so that it teaches these new concepts in a new way. Fourth, we created a set of short videos that explain key concepts and walk the reader through UML diagrams to help with understanding complex modeling.

For example, Chapter 1 presents a complete iteration in the development of a new system. Students get to see that complete iteration—from beginning to end (through implementation and testing)—before having to learn abstract

principles or memorize terms. Also, the newly written running cases throughout the book focus on current issues of communication and connectedness and take the students through all aspects of system development. We have also expanded the *Instructor's Materials* and enhanced the aids available through CourseMate, our online resource. Additional online chapters are also available to enhance and extend the learning experience.

Finally, we updated and enhanced the set of *over 30 short videos* that explain key concepts in the text. These videos have been very well received and are even better with the new edition. These videos are useful for blended and online classes as well as traditional classes. The videos range from 3 to 10 minutes, and provide just-in-time explanations for often difficult to understand concepts, such as iterative development and Agile development, and illustrate important techniques such as identifying user stories and use cases. Most importantly, the videos show by demonstration how to read and interpret important UML models such as the domain model class diagrams, use case diagrams, sequence diagrams, and package diagrams. Understanding detailed UML models is finally possible in a way no other text can match.

We are excited about this new approach. The time is right for new materials and new tools for teaching systems analysis and design. Instructors will find this textbook intuitive, powerful, and easy to use. Students will find it engaging and empowering. Many concepts are presented so the students can teach themselves, with coaching and direction provided by the professor. It will be an rewarding experience to teach and learn with this textbook.

■ Innovations

This edition is innovative in many respects. It includes key concepts from traditional and object-oriented approaches, covers the use case-driven approach (with UML modeling being detailed in depth), emphasizes Agile and iterative development, and incorporates the latest concepts in Agile project management. Also, the material is completely reorganized to better support learning systems analysis and design.

■ Coverage of Object Orientation and Traditional Analysis and Design

This textbook is unique in its integration of key systems-modeling concepts that apply to the traditional structured approach and the object-oriented approach—user goals and events that trigger system use cases, plus classes of objects/data entities that are part of the system's problem domain. We devote one chapter to identifying user stories and use cases and another chapter to modeling key objects/entities, including coverage of entity-relationship diagrams, while emphasizing UML domain model class diagrams. After completing these chapters, instructors can cover structured analysis and design by including an online chapter, or they can focus on object-oriented analysis and design by using the chapters in this textbook. It is assumed from the beginning that everyone should understand the key object-oriented concepts. The traditional approach isn't discarded; key structured concepts are still included. But these days, most instructors are emphasizing the object-oriented approach.

■ Full Coverage of UML and the Object-Oriented Approach

The object-oriented approach presented in this textbook is based on the Unified Modeling Language (UML 2.0) from the Object Management Group, as originated by Grady Booch, James Rumbaugh, and Ivar Jacobson. A model-driven approach to analysis starts with user stories and use cases and then defines problem domain classes involved in the users' work. We include requirements

modeling with use case diagrams, domain modeling, use case descriptions, activity diagrams, and system sequence diagrams. The FURPS+ model is used to emphasize functional and nonfunctional requirements.

Design principles and design patterns are discussed in depth, and system architecture is modeled by using UML component diagrams and package diagrams. Detailed design models are also discussed in detail, with particular attention given to use case realization with CRC cards, sequence diagrams, and design class diagrams.

■ Project Management Coverage

Many undergraduate programs depend on their systems analysis and design course to teach project management principles. To satisfy this need, we cover project management by taking a four-pronged approach. First, specific project management techniques, skills, and tasks are included and highlighted throughout this book. This integration teaches students how to apply specific project management tasks to the various activities of the system development life cycle (SDLC), including iterative development. Second, complete coverage of project planning and project management is included in a separate chapter. Third, we include a 120-day trial version of Microsoft Project Professional in the back of this book so students can obtain hands-on experience with this important tool. Fourth, a more in-depth treatment of project management techniques and principles is provided in an online chapter on this book's Web site. This information is based on the Project Management Body of Knowledge (PMBOK), as developed by the Project Management Institute—the primary professional organization for project managers in the United States.

■ Organized for More Effective Learning

This edition's innovative and entirely new organization starts with a complete beginning-to-end example of system development, moves immediately to systems analysis models and techniques, and then proceeds to system design concepts, emphasizing system architecture, user interfaces, and database design. The student sees analysis and much of design covered in the first nine chapters. Next, the text focuses on managing system development projects, including Agile development, after the student has had a chance to understand what is actually involved in system development. Finally, the text covers detailed design topics and deployment topics, going into more depth about such contemporary approaches as the Unified Process, Extreme Programming, and Scrum.

■ CourseMate Companion Web Site

Cengage Learning's *Systems Analysis and Design in a Changing World, Seventh Edition*, CourseMate brings course concepts to life with interactive learning, study, and exam preparation tools that support the printed textbook. Watch student comprehension soar as your class works with the printed textbook and the textbook-specific Web site. CourseMate goes beyond the book to deliver what you need! Learn more at cengage.com/coursemate.

■ Engagement Tracker

How do you assess your students' engagement in your course? How do you know your students have read the material or viewed the resources you have assigned? How can you tell if your students are struggling with a concept? With CourseMate, you can use the included Engagement Tracker to assess student preparation and engagement. Use the tracking tools to see progress for the class as a whole or for individual students. Identify students at risk early in the course. Uncover which concepts are most difficult for your class. Monitor time on task. Keep your students engaged.

■ Interactive Teaching and Learning Tools

CourseMate includes interactive teaching and learning tools:

- Quizzes
- Case projects
- Flash cards
- Short videos on concepts, techniques, and models
- PowerPoint presentations

These assets enable students to review for tests, prepare for class, and address the needs of students' varied learning styles.

■ Interactive E-Book

In addition to interactive teaching and learning tools, CourseMate includes an interactive e-book. Students can take notes, highlight, search for, and interact with embedded media specific to their book. Use it as a supplement to the printed text or as a substitute—the choice is your students' with CourseMate.

■ Organization and Use

Systems Analysis and Design in a Changing World, Seventh Edition, includes this printed textbook, a complete e-book, and supporting online chapters. The current printed textbook provides a focused presentation of those topics that are essential and most important for information systems developers. The online chapters extend those concepts and provide a broader presentation of several topics. The online chapters may be integrated into the course or simply used as additional readings as prescribed by the instructor.

There are three major subject areas discussed in this book: systems analysis, systems design, and project management. There are additional subject areas, which are no less important but aren't discussed in as much depth. These include systems implementation, testing, and deployment. In addition, we have taken an approach that is quite different from other texts. Because students already have a basic understanding of systems analysis and design from Chapter 1, we immediately present in-depth concepts related to systems analysis and design. We present approaches to development and project management topics later in the text. This allows students to learn those project management concepts after understanding the elements of systems analysis and design. We think it will be more meaningful for students at that point in the course.

■ Part 1: Introduction to System Development

Part 1, comprising Online Chapter A and Chapter 1, presents an overview of system development. Online Chapter A, "The Role of the Systems Analyst," describes basic systems concepts and the role of the systems analyst in system development projects. Chapter 1 begins by briefly explaining the objectives of systems analysis and systems design. Then, it provides a detailed, concrete example of what is required in a typical software development project. Many students who take a programming class think that programming is all you need to develop software and deploy a system. This chapter and the rest of the book should dispel that myth.

■ Part 2: Systems Analysis Tasks

Chapters 2 through 5 cover systems analysis in detail. Chapter 2 discusses system requirements, analysis activities, and techniques for gathering information about the business problem. Developing the right system solution is possible only

if the problem is accurately understood. Chapter 2 also explains how to identify and involve the stakeholders and introduces the concept of models and modeling. Chapters 3 and 4 teach modeling techniques for capturing the detailed requirements for the system in a useful form. When discussing an information system, two key concepts are particularly useful: the user stories/use cases that define what the end users need the system to do and the data entities/domain classes that users work with while carrying out their work tasks. These two concepts—user stories/use cases and data entities/domain classes—are important no matter what approach to system development is being used. Chapter 5 presents more in-depth requirements models, such as use case descriptions, activity diagrams, system sequence diagrams, and CRUD analysis.

Online Chapter B, “The Traditional Approach to Requirements,” presents the traditional, structured approach to developing systems. To those instructors and students who desire to learn about data flow diagrams and structured English, this chapter provides an in-depth presentation.

All these modeling techniques provide in-depth analysis of user needs and allow the analyst to develop requirements and specifications. Again, the purpose of systems analysis is to thoroughly understand and specify the user’s needs and requirements.

■ **Part 3: Essentials of Systems Design**

Part 3 provides the fundamental concepts related to systems design and designing the user experience. Chapter 6 provides broad and comprehensive coverage of important principles of systems design, including design activities and the crucial issues of system controls and security that all students should understand. It serves not only as a broad overview of design principles but also as a foundation for later chapters that explain the detailed techniques, tasks, skills, and models used to carry out design.

Chapter 7 provides a comprehensive overview of system architecture and is a new chapter that consolidates material previously spread out in multiple chapters. Chapter 8 presents additional design principles related to the user experience. Designing the user interface is a combination of analysis and design. It is related to analysis because it requires heavy user involvement and includes specifying user activities and desires. On the other hand, it is a design activity because it is creating specific final components that are used to drive the programming effort. The screens and reports and other user interaction components must be precisely designed so they can be programmed as part of the final system. Chapter 9 provides a compact and integrated coverage of designing the database.

■ **Part 4: Projects and Project Management**

By this point, students will have a basic understanding of all the elements of system development. Part 4 brings together all these concepts by explaining more about the process of organizing and managing development projects. Chapter 10 describes different approaches to system development in today’s environment, including Agile development and several widely used development methodologies—the Unified Process, Extreme Programming, and Scrum. It is an important chapter to help you understand how projects actually get executed.

Chapter 11 extends these concepts by teaching foundation principles of project planning and project management. Every systems analyst is involved in helping organize, coordinate, and manage software development projects. In addition, most good students will eventually become team leaders and project managers. The principles presented in Chapter 11 are essential to a successful career.

Online Chapter C, “Project Management Techniques,” goes into more detail regarding the tools and techniques used by systems analysts and project managers to plan and monitor development projects. For those instructors and students who would like to learn specific project management skills, this is an important chapter.

■ **Part 5: Advanced Design and Deployment Concepts**

Part 5 goes into more depth with respect to systems design, specifically object-oriented software design, and other important issues related to effective and successful system development and deployment.

Chapters 12 and 13 explain in detail the models, skills, and techniques used to design software systems. As mentioned earlier, systems design is a fairly complex activity, especially if it is done correctly. The objective of these two chapters is to teach the student the various techniques—from simple to complex—that can be used to effectively design software systems.

Chapter 14 describes the final elements in system development: final testing, deployment, maintenance, and version control.

■ **Designing Your Analysis and Design Course**

There are many approaches to teaching analysis and design courses, and the objectives of the course differ considerably from college to college. In some academic information systems departments, the analysis and design course is a capstone course in which students apply the material learned in prior database, networking, and programming courses to a real analysis and design project. In other information systems departments, analysis and design is used as an introduction to the field of system development and is taken prior to more specialized courses. Some information systems departments offer a two-course sequence emphasizing analysis in the first semester and design and implementation in the second semester. Some information systems departments have only one course that covers analysis and design.

The design of the analysis and design course is complicated even more by the choice of emphasizing some traditional and some object-oriented content—again, depending on local curriculum priorities. Additionally, the more iterative approach to development in general has made choices about sequencing the analysis and design topics more difficult. For example, with iterative development, a two-course sequence can’t be divided into analysis and then design as easily.

The objectives, course content, assignments, and projects have many variations. What we offer below are some suggestions for using this textbook in various approaches to the course.

■ **UML and Object-Oriented Analysis and Design Course**

This is the course we designed the printed textbook to support, so all the printed chapters but none of the online chapters are included. Note that object-oriented design is included in detail. The course covers object-oriented analysis and design, user and system interface design, database design, controls and security, and implementation and testing. It is usually assumed that the projects will use custom development, including Web development. The course emphasizes iterative development with three-layer architecture, project management, information gathering, and management reporting. One-semester courses are usually limited to completing some prototypes of the user interface to give students closure. Sometimes, this course is spread over two semesters, with some implementation of an actual system in the second semester for a more complete development experience. Iterative development is emphasized.

A suggested outline for a course emphasizing object-oriented development is:

Online Chapter A: The Role of the Systems Analyst (optional)
 Chapter 1: From Beginning to End: An Overview of Systems Analysis and Design
 Chapter 2: Investigating System Requirements
 Chapter 3: Identifying User Stories and Use Cases
 Chapter 4: Domain Modeling
 Chapter 5: Use Case Modeling
 Chapter 6: Foundations for Systems Design
 Chapter 7: Defining the System Architecture
 Chapter 8: Designing the User Interface
 Chapter 9: Designing the Database
 Chapter 10: Approaches to System Development
 Chapter 11: Project Planning and Project Management
 Chapter 12: Object-Oriented Design: Fundamentals
 Chapter 13: Object-Oriented Design: Use Case Realization
 Chapter 14: Deploying the New System

■ Traditional Analysis and Design Course

A traditional systems analysis and design course provides coverage of activities and tasks by using structured analysis, user and system interface design, database design, controls and security, and implementation and testing. It is usually assumed that the project will use custom development, including Web development. The course emphasizes the SDLC, project management, information gathering, and management reporting. One-semester courses are usually limited to completing some prototypes of the user interface to give students closure. Sometimes, this course is spread over two semesters, with some implementation of an actual system in the second semester for a more complete development experience.

For this approach to the analysis and design course, a reasonable outline would omit chapters and sections detailing object orientation but include the online chapters on the role of the systems analyst and on traditional structured analysis. However, object-oriented concepts are introduced throughout the text, so students will still be familiar with them. Additionally, because of the amount of material to cover, the online chapter detailing project management, financial feasibility, and scheduling might be omitted.

A suggested outline for a course emphasizing the traditional structured approach is:

Online Chapter A: The Role of the Systems Analyst
 Chapter 1: From Beginning to End: An Overview of Systems Analysis and Design
 Chapter 2: Investigating System Requirements
 Chapter 3: Identifying User Stories and Use Cases
 Chapter 4: Domain Modeling
 Online Chapter B: The Traditional Approach to Requirements
 Chapter 6: Foundations for Systems Design
 Chapter 8: Designing the User Interface
 Chapter 9: Designing the Database
 Chapter 10: Approaches to System Development
 Chapter 11: Project Planning and Project Management
 Chapter 14: Deploying the New System

■ In-Depth Analysis and Project Management

Some courses cover object-oriented systems analysis methods in more depth and briefly survey structured analysis—with not much about object-oriented

design—while emphasizing project management. Sometimes, these courses are graduate courses; sometimes, they assume design and implementation are covered in more technical courses. In some cases, it might be assumed that packages are likely solutions rather than custom development, so defining requirements and managing the process are more important than design activities. The online chapters covering the role of the systems analyst, the traditional approach to structured analysis, and project management would be included.

A suggested outline for a course emphasizing object-oriented analysis, with in-depth coverage of project management, is:

Online Chapter A: The Role of the Systems Analyst

Chapter 1: From Beginning to End: An Overview of Systems Analysis and Design

Chapter 2: Investigating System Requirements

Chapter 3: Identifying User Stories and Use Cases

Chapter 4: Domain Modeling

Chapter 5: Use Case Modeling

Online Chapter B: The Traditional Approach to Requirements

Chapter 6: Foundations for Systems Design

Chapter 8: Designing the User Interface

Chapter 10: Approaches to System Development

Chapter 11: Project Planning and Project Management

Online Chapter C: Project Management Techniques

Chapter 14: Deploying the New System

■ Available Support

Systems Analysis and Design in a Changing World, Seventh Edition, includes teaching tools to support instructors in the classroom. The ancillary materials that accompany the textbook include an *Instructor's Manual*, solutions, test banks and test engine, PowerPoint presentations, and figure files. Please contact your Cengage Course Technology sales representative to request the Teaching Tools CD-ROM if you haven't already received it. Or go to the Web page for this book at login.cengage.com to download all these items.

■ The Instructor's Manual

The *Instructor's Manual* includes suggestions and strategies for using the text, including course outlines for instructors that emphasize the traditional structured approach or the object-oriented approach. The manual is also helpful for those teaching graduate courses on analysis and design.

■ Solutions

We provide instructors with answers to review questions and suggested solutions to chapter exercises and cases. Detailed traditional and UML object-oriented models are included for all exercises and cases that ask for modeling solutions.

■ ExamView

This objective-based test generator lets the instructor create paper, LAN, or Web-based tests from test banks designed specifically for this Course Technology text. Instructors can use the QuickTest Wizard to create tests in fewer than five minutes by taking advantage of Course Technology's question banks or instructors can create customized exams.

■ WebTUTOR™ Plug and Play!

Jump-start your course with customizable, text-specific content within your Course Management System!

- **Jump-start**—Instructors simply load a WebTutor cartridge or e-Pack into their Course Management System.
- **Content**—Students have access to text-specific content, media assets, quizzing, Web links, discussion topics, interactive games and exercises, and more.
- **Customizable**—Instructors can easily blend, add, edit, reorganize, or delete content.

Whether you want to Web-enable your class or put an entire course online, WebTutor delivers! Visit academic.cengage.com/webtutor to learn more.

■ Product Description

WebTutor and WebTutor Toolbox products are Course Cartridges and e-Packs that provide content natively on a Course Management System (WebCT, BlackBoard, Angel, D2L, and eCollege). The purpose of the product is to provide electronic solutions in an easy-to-use format with little up-front costs to instructors.

- For more information on how to bring WebTutor to your course, instructors should contact their Cengage Learning sales representative.

■ PowerPoint Presentations

Microsoft PowerPoint slides are included for each chapter. Instructors might use the slides in a variety of ways, such as teaching aids during classroom presentations or as printed handouts for classroom distribution. Instructors can add their own slides for additional topics they introduce to the class.

■ Figure Files

Figure files allow instructors to create their own presentations by using figures taken directly from this text.

■ Credits and Acknowledgments

We have been very gratified as authors to receive so many supportive and enthusiastic comments about *Systems Analysis and Design in a Changing World*. Students and instructors in the United States and Canada have found our text to be the most up-to-date and flexible book available. The book has also been translated into many languages and is now used productively in Europe, Australia, New Zealand, India, China, and elsewhere. We truly thank everyone who has been involved in all the editions of our textbook, particularly Lori Bradshaw who managed the development of the seventh edition.

We also want to thank all the reviewers who worked so hard for us—beginning with an initial proposal and continuing throughout the completion of all seven editions of this text. We were lucky enough to have reviewers with broad perspectives, in-depth knowledge, and diverse preferences. We listened very carefully, and the text is much better as a result of their input. Reviewers for the various editions include:

Rob Anson, *Boise State University*
 Marsha Baddeley, *Niagara College*
 Teri Barnes, *DeVry Institute—Phoenix*
 Robert Beatty, *University of Wisconsin—Milwaukee*

James Buck, *Gateway Technical College*
Anthony Cameron, *Fayetteville Technical Community College*
Genard Catalano, *Columbia College*
Paul H. Cheney, *University of Central Florida*
Kim Church, *Oklahoma State University*
Jung Choi, *Wright State University*
Jon D. Clark, *Colorado State University*
Mohammad Dadashzadeh, *Oakland University*
Lawrence E. Domine, *Milwaukee Area Technical College*
Gary Garrison, *Belmont University*
Cheryl Grimmett, *Wallace State Community College*
Jeff Hedrington, *University of Phoenix*
Janet Helwig, *Dominican University*
Susantha Herath, *St. Cloud State University*
Barbara Hewitt, *Texas A&M University*
Ellen D. Hoadley, *Loyola College in Maryland*
Jon Jaspersen, *Texas A&M University*
Norman Jobses, *Conestoga College—Waterloo, Ontario*
Gerald Karush, *Southern New Hampshire University*
Robert Keim, *Arizona State University*
Michael Kelly, *Community College of Rhode Island*
Rajiv Kishore, *The State University of New York—Buffalo*
Rebecca Koop, *Wright State University*
Hsiang-Jui Kung, *Georgia Southern University*
James E. LaBarre, *University of Wisconsin—Eau Claire*
Ingyu Lee, *Troy University*
Terrence Linkletter, *Central Washington University*
Tsun-Yin Law, *Seneca College*
David Little, *High Point University*
George M. Marakas, *Indiana University*
Roger McHaney, *Kansas State University*
Cindi A. Nadelman, *New England College*
Bruce Neubauer, *Pittsburgh State University*
Michael Nicholas, *Davenport University—Grand Rapids*
Mary Prescott, *University of South Florida*
Alex Ramirez, *Carleton University*
Eliot Rich, *The State University of New York—Albany*
Robert Saldarini, *Bergen Community College*
Laurie Schatzberg, *University of New Mexico*
Deborah Stockbridge, *Quincy College*
Jean Smith, *Technical College of the Lowcountry*
Peter Tarasewich, *Northeastern University*
Craig VanLengen, *Northern Arizona University*
Bruce Vanstone, *Bond University*
Haibo Wang, *Texas A&M University*
Terence M. Waterman, *Golden Gate University*



Introduction to System Development

PART ONE

- ▶ **Online Chapter A**
The Role of the Systems Analyst
- ▶ **Chapter 1**
From Beginning to End:
An Overview of Systems Analysis
and Design



From Beginning to End: An Overview of Systems Analysis and Design

CHAPTER ONE

CHAPTER OUTLINE

- Software Development and Systems Analysis and Design
- The System Development Life Cycle (SDLC)
- Iterative Development
- Introduction to Ridgeline Mountain Outfitters (RMO)
- Developing RMO's Tradeshaw System
- Where You Are Headed—The Rest of This Book

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- Describe the purpose of systems analysis and design when developing information systems
- Explain the purpose of the system development life cycle (SDLC) and identify its six core processes
- Explain how information system methodologies provide guidelines for completing the six core processes of the SDLC
- Describe the characteristics of Agile methodologies and iterative system development
- Based on the Ridgeline Mountain Outfitters Tradeshaw System example:
 - Describe how the six core processes of the SDLC are used in each iteration
 - Identify key documents used in planning a project
 - Identify key diagrams used in systems analysis and systems design

■ Software Development and Systems Analysis and Design

You have grown up in a world of ubiquitous computing, where computers are everywhere and are increasingly characterized by mobility, communication, and connectivity. You use smartphones, laptops, notepads, and wearable devices throughout the day. Some of you have already developed your own application software or have friends who have written applications for these devices. Some of you have taken programming classes; others have taught yourself how to write computer application programs. In one way or another, you are certainly interested in building computer applications and information systems.

Although you are most likely more familiar with your mobile devices, there is much more to building information systems than just that. Information systems exist to support all aspects of business organizations and have done so for centuries. The ancient Mesopotamians conducted business and had accounting information systems 3,000 years ago—using clay tablet technology. Electronic computers have been a part of these information systems only for the last 50 years. The technology changes, but information systems have a long history.

information system a set of interrelated components that collect, process, store, and provide as output the information needed to complete business tasks

An **information system** is a set of interrelated components that collect, process, store, and provide as output the information needed to complete business tasks. The information system always includes people who operate the system and carry out some of the work. In Mesopotamia, people did just about all of the work required. Now, of course, electronic computing devices do most of the work, although not all. If you are at the library typing in some search terms using the online catalog, you are part of the information system—the part that supplies the input and consumes the output. If you are using your bank’s online information system, you are part of the information system—the part that selects which account to use to pay a specific bill.

computer application or app a computer software program that executes on a computing device to carry out a specific function or set of related functions

More recently, another term has been used to refer to an information system—a computer application. A **computer application** is a computer software program that executes on a computing device to carry out a specific function or set of related functions. Sometimes, *computer application* is shortened to **app** (such as an iPhone app or an Android app). Many people use the terms information system and computer application interchangeably, but remember that an information system includes people and their manual procedures and an application usually refers just to the software.

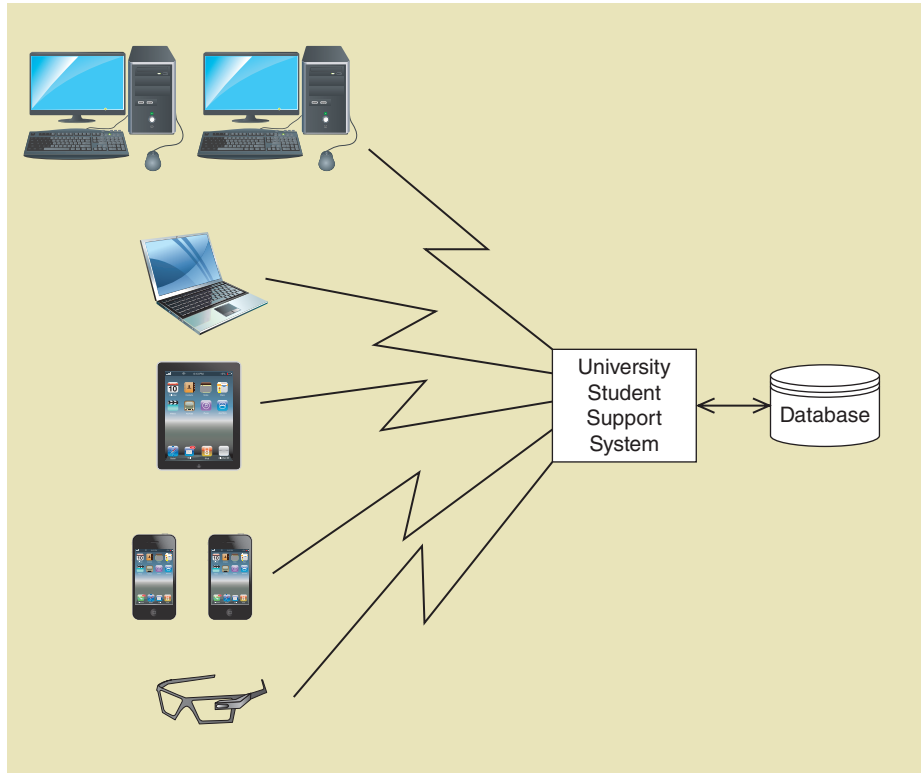
Consider the information system your university or college uses to support students. It is an elaborate system that likely integrates admissions, financial aid, course scheduling, and even individual course support. You probably access this information system through the network using a desktop workstation at home or in a computer lab, a wireless notebook computer, an iPad or tablet, an iPhone or an Android phone, and even a wearable device such as a smartwatch or Google Glass. There might be an app that connects to the system seamlessly from your device, or you might connect through a browser on your desktop, notebook, or other devices. **Figure 1-1** shows a variety of devices all connecting to the same University Student Support System.

Each information system (or app) was conceived and built to satisfy some need. When the information system is completed, it is used productively to satisfy that need. Our purpose here is to describe the process by which an information system is created from perceived need through actual use. As noted in this chapter’s title, systems analysis and systems design are key components of this process.

systems analysis those system development activities that enable a person to understand and specify what the new system should accomplish

Systems analysis consists of those activities that enable a person to understand and specify what the new system should accomplish. The operative words here are *understanding* and *specifying*. Systems analysis is more than a brief

FIGURE 1-1 A variety of devices all connected to the same information system



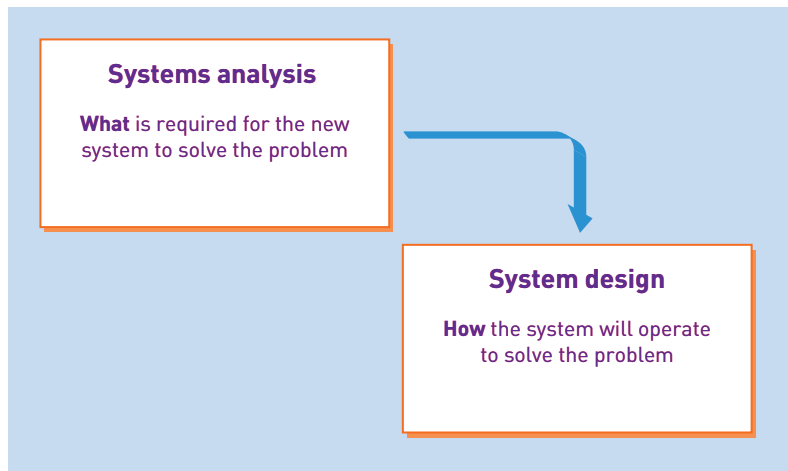
systems design those system development activities that enable a person to describe in detail how the resulting information system will actually be implemented

statement of the problem. For example, a customer management system must track customers, register products, monitor warranties, and track service levels, among many other functions—all of which have many details. Systems analysis describes in detail *what* a system must do to satisfy the need or solve the problem.

Systems design consists of those activities that enable a person to describe in detail how the information system will actually be implemented to provide the needed solution. In other words, systems design describes *how* the system will actually work. It specifies in detail all the components of the solution system and how they work together. See **Figure 1-2** to help distinguish between analysis and design.

Systems analysis and design plays an integral role in the development of information systems. To illustrate, consider an analogous situation; the art and

FIGURE 1-2 Systems analysis versus systems design



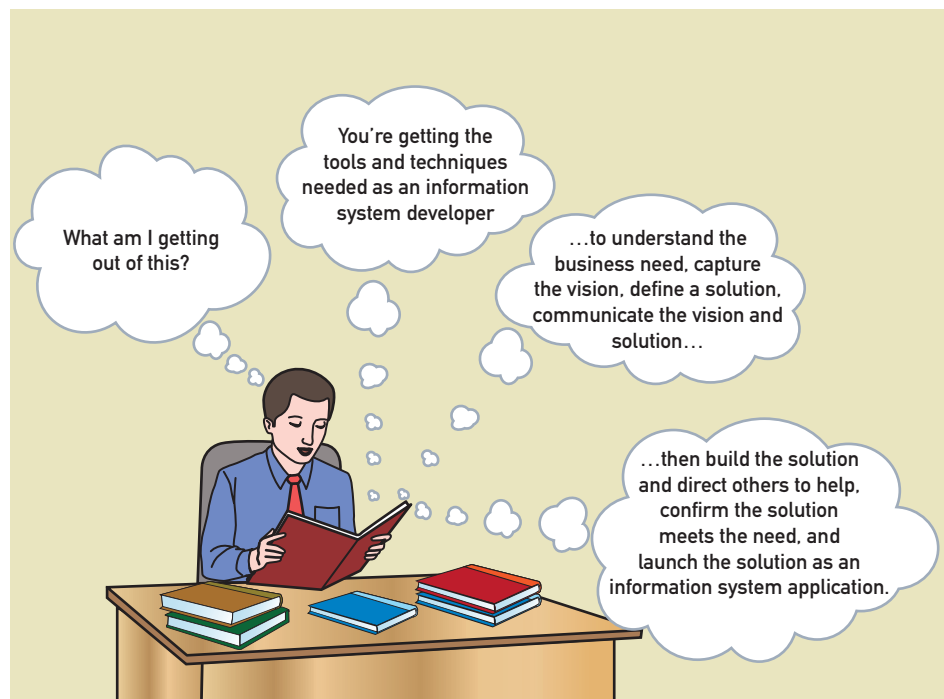
science of creating a new building. In this scenario, there is the owner of the land who has the vision, the builder who will construct the building, and the architect who serves as the bridge between the owner and the builder. The architect helps the owner develop the vision, but must also communicate the building's specifications to the builder. In doing so, the architect uses various tools to first capture the vision from the owner and to then provide the builder with instructions—including line drawings, blueprints, to-scale models, detail specifications, and even on-site inspection reports.

Just as a builder doesn't start construction without plans, programmers don't just sit down and start writing program code. They need someone (maybe themselves) to function like an architect—planning, capturing the vision, understanding details, specifying needs—before designing and writing the code that satisfies the vision. Usually, we call this person a *systems analyst*. In situations where you are the programmer as well as the analyst (often called a *programmer-analyst*), it might be possible to keep track of the details without many formal notes. However, in today's world, with system development teams often distributed worldwide, you might only be responsible for part of the programming, with the rest handled by team members in different locations. In a distributed team situation or with a complicated project, it is much more important to create formal requirements documents that capture each component's specifications.

In a nutshell, systems analysis and design provides the tools and techniques you need as an information system developer to complete the development process:

1. Understand the need (business need).
2. Capture the vision.
3. Define a solution.
4. Communicate the vision and the solution.
5. Build the solution or direct others in building the solution.
6. Confirm that the solution meets the need.
7. Launch the solution application.

FIGURE 1-3 What analysis and design provides for the system developer



© Cengage Learning®

■ The System Development Life Cycle (SDLC)

project a planned undertaking that has a beginning and an end and produces some end result

system development life cycle (SDLC) a framework that identifies all the activities required to research, build, deploy, and often maintain an information system

system development process or **methodology** a set of comprehensive guidelines for carrying out all of the activities of each core process of the SDLC

Initial development of a new information system is usually done as a project. A **project** is a planned undertaking that has a beginning and an end and produces some end result. This means that the activities required to develop a new system are identified, planned, organized, and monitored. Some projects are very formal, whereas others are informal, usually depending on the project size.

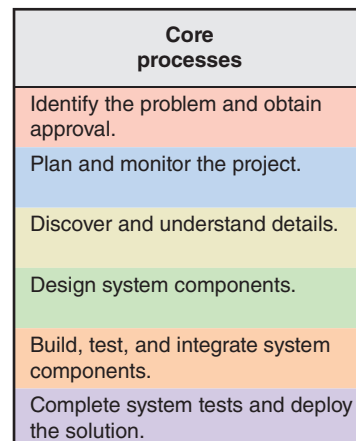
To manage a project with analysis, design, and other development activities, you need a project management framework to guide and coordinate the work of the project team. The **system development life cycle (SDLC)** is a framework that identifies all the activities required to research, build, deploy, and often maintain an information system. Normally, the SDLC includes all activities needed for the planning, systems analysis, systems design, programming, testing, and user training stages of information systems development, as well as other project management activities that are required to successfully deploy the new information system.

There are many approaches to the SDLC, including variations specific to certain types of projects. However, every SDLC includes some core processes that are always required, though many different names are used. Here are the six core processes required in the development of any information system (see **Figure 1-4**):

- Identify the problem or need and obtain approval to proceed with the project.
- Plan and monitor the project—what to do, how to do it, and who does it.
- Discover and understand the details of the problem or the need—what is required?
- Design the system components that solve the problem or satisfy the need—how will it actually work?
- Build, test, and integrate system components—lots of programming and component integration.
- Complete system tests and then deploy the solution—the need now is satisfied.

As previously stated, most information systems you will develop are conceived and built to solve complex organizational problems, which are usually very complex, thus making it difficult to plan and manage a system development project. Fortunately, there are many ways to implement the six core processes of the SDLC to handle each project's complexity. An information systems development **methodology** is a set of comprehensive guidelines for carrying out all of the activities of each core process of the SDLC. An overall **system development process** is a more recent term for methodology. Each development

FIGURE 1-4 Six core processes of the SDLC



© Cengage Learning®

Agile development an information system development process that emphasizes flexibility and rapid response to anticipate new and changing requirements during development

methodology prescribes a way of carrying out the development project, and every organization develops its own system development methodology over time to suit its needs.

During the last 15 years, information system research efforts have resulted in many new information systems development methodologies/processes to improve the chance of project success. These are all based on what is called **Agile development**. The basic philosophy of Agile development is that neither team members nor the users completely understand the problems and complexities of a new system, so the project plan and the execution of the project must be responsive to unanticipated issues. The plan must be agile and flexible. It must have procedures in place to allow for, anticipate, and even embrace changes and new requirements that come up during the development process. The six core processes are still involved in Agile development, but they are carried out iteratively, as explained next.

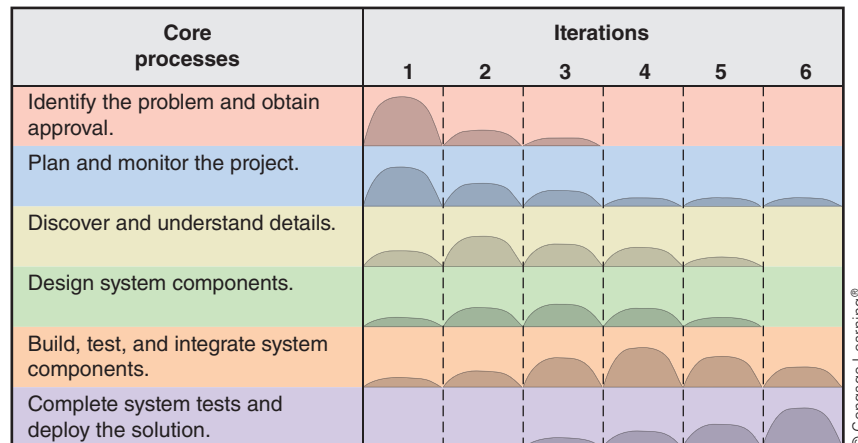
■ Iterative Development

Iterative development an approach to system development in which the system is “grown” piece by piece through multiple mini-projects called iterations

Iterative development is an approach to system development in which the system is “grown” in an almost organic fashion. Core components are developed first and then additional components are added. It is called *iterative* because the six core development processes are repeated for each component. In other words, there is one big project, which consists of a series of mini-projects, and the information system is grown piece by piece during these mini-projects. Iterative development makes Agile development possible, although Agile development includes additional techniques that help with project flexibility, too.

Figure 1-5 illustrates how an iterative project might be managed. Across the figure, you see six iterations as columns. Each iteration involves all six core processes, shown as rows in the table. At the end of each iteration, a working part of the system is completed and evaluated. An iteration lasts a fixed period of time, usually two to four weeks. The rounded mounds inside the graph represent the relative amount of effort for that core process during that iteration. For example, in Figure 1–5, Iteration 1 appears to primarily focus on identifying the problem and planning the project. Lesser amounts of discovery, design, and build/test may also be done. In Iteration 2, there is less effort for identifying the problem and planning the project and more effort for discovery, design, and build/test. By Iteration 3, build/test gets the most effort, but all six core processes are still involved, including the beginnings of completing and deploying the system.

FIGURE 1-5 The six core processes of the SDLC showing iterations



There are several benefits to iterative development. For one, portions of the system can sometimes be deployed sooner. If there are core functions that provide basic support for users, these can be deployed in an early iteration. Second, by taking a small portion and developing it first, the most difficult problems can be identified and addressed early in the project. Many of today's systems are so large and complex that even with a formal process it is impossible to remember and understand everything. By focusing on only a small portion at a time, the requirements are fewer and easier to solve. Finally, developing a system in iterations makes the entire development process more flexible and able to address new requirements and issues that come up throughout the project.

A key element of iterative development is dividing system components into pieces that can be completed in two to four weeks. During one iteration, all the core development processes are involved, including programming and system-wide testing, so the result is a working part of the system, even though it may only have a portion of the functionality that is ultimately required. Developers choose components for each iteration based on priority, either the components most needed or riskiest to implement.

To better illustrate these concepts, we will walk through a complete example in the next sections concerning Ridgeline Mountain Outfitters (RMO). These sections use a fairly small information system to demonstrate all six core processes (as much as is feasible in a textbook, anyway). The example completes one iteration in detail, but the project actually requires multiple iterations. By going all the way through a very simple project, you will more easily understand the complex concepts provided in the rest of the text.

■ Introduction to Ridgeline Mountain Outfitters (RMO)

Ridgeline Mountain Outfitters (RMO) is a large retail company that specializes in clothing and related accessories for all types of outdoor and sporting activities. The mountain and western regions of the United States and Canada witnessed tremendous growth in recreation activities in recent years, including skiing, snowboarding, mountain biking, water skiing, jet skiing, river running, sailing, jogging, hiking, ATVing, cycling, camping, mountain climbing, and rappelling. With the increased interest in outdoor sports, the market for winter and summer sports clothing and accessories exploded, so RMO continually expanded its line of sportswear to respond to this market.

The company's growth charted an interesting history of mail-order, brick-and-mortar, and online sales. RMO got its start by selling to clothing stores in the Park City, Utah, area. In the late 1980s and early 1990s, it began selling directly to customers using catalogs with mail-in and telephone-order options. It opened its first store in 1994. After the Winter Olympics in Park City in 2002, business exploded and RMO quickly expanded to 10 retail outlets throughout the West and added Internet sales. Last year, retail store revenue was \$67 million, telephone- and mail-order revenues were \$10 million, and Internet sales were \$200 million. Most sales continue to be in the West, although the market in several areas of the eastern United States and Canada is growing. By the Winter Olympics in Vancouver, British Columbia, in 2008, RMO's growth and profits resulted mainly from online sales and service, as with most specialty retailers; however, the brick-and-mortar and mail-order business remained important, too. After the Winter Olympics in Sochi in 2014, RMO negotiated with several Utah Olympic Medal Winners for endorsements. This provided additional interest throughout the West and instigated another period of rapid growth.

Figure 1-6 shows a sample of the catalog that RMO still mails out. Although mail-order and telephone sales are modest, receiving the catalog encourages

FIGURE 1-6 RMO winter catalog

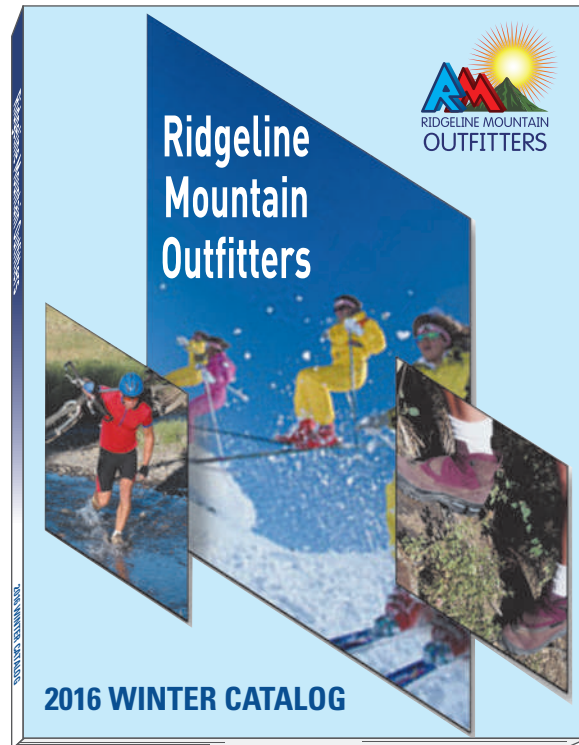
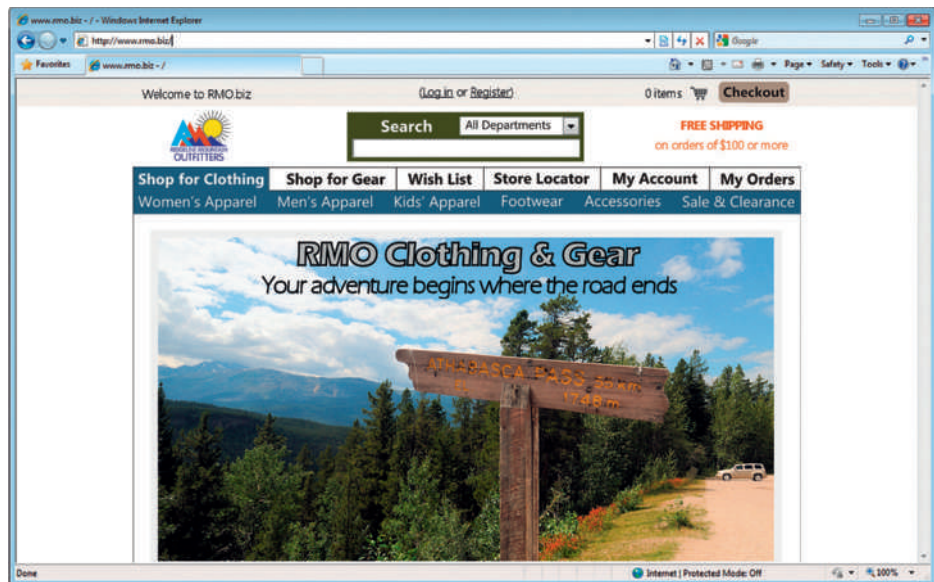


FIGURE 1-7 RMO online ordering home page



customers to go online to make purchases, so RMO continues to produce and mail abbreviated versions. **Figure 1-7** shows the RMO online ordering home page.

RMO produces its own line of outdoor clothing and sportswear. However, to offer a complete range of clothing in its retail outlets, it also sells brands of clothing sourced from other vendors. Furthermore, most accessories sold are sourced through vendors.

■ Trade Shows

To keep its product line innovative and responsive to consumer demand, RMO's purchasing agents attend apparel and accessory trade shows around the world

where vendors exhibit their merchandise. RMO is good at anticipating trends and profiting from interesting vendor specials. Furthermore, its agents are always watching for new products and accessories to expand RMO's product line appropriately.

At the trade shows, RMO purchasing agents frequently find products they want to add to the spring, summer, or winter apparel offering. In the past, when RMO purchasing agents wanted to place an order with a vendor, they would exchange contact information with the vendor at the trade show and would then follow up via e-mails and phone calls to create a purchase order. In the current 24/7 business climate, this business process was just too slow. RMO needed to speed things up to keep ahead of the competition, take advantage of vendor deals at the trade shows, and be more responsive to customer demands.

To solve this problem, RMO initiated an information system project to develop a system for collecting and tracking information about suppliers and about products added to its merchandise offerings. The Tradeshow System needs to take advantage of the latest in wireless devices and data capturing technology to allow purchasing agents to research and complete purchase orders on the spot at the trade shows. RMO decided to use an agile, iterative project management approach to get the small system completed as fast as possible with maximum flexibility.

■ Developing RMO's Tradeshow System

We will organize our sample project—the RMO Tradeshow System—into several iterations. Our plan for the first iteration is to have it finished in just six days. Our primary objective is to introduce you to the concepts and techniques of the six core processes. To do this, we may go a little deeper into a core process than we might usually do on the first iteration of a real project. Additionally, the iteration will appear to be managed much more formally than might be the case in the real world for such a small project. The second and subsequent iterations will not be described in any detail, but the complete Tradeshow System project will need several more iterations for a finished product.

Most new information system applications require a project with several iterations. In the first iteration, there are usually three major objectives. The first objective is to get project approval. The second objective is to get a clear picture of the system's overall vision—the overall functions and data requirements. The third objective is to determine the detail specifications and develop a solution for one portion of the system (i.e., actually analyze, design, build, and test one part of the system). The second and third iterations would continue to work on the additional portions of the system based on the system vision.

In our project, we will touch on all these objectives within the first iteration. We will show an example of a System Vision Document and then develop one portion of the overall system. It should be noted that the division of this project into days and daily activities is somewhat arbitrary. The following organization is quite workable, but it is not the only way to organize the project.

■ Initial Project Activities

Before the project actually begins, the head of RMO's Purchasing Department works with a systems analyst to identify and document the specific business need and to define the project objectives. RMO's management then reviews the primary project objectives and provides budget approval. Every organization has to give budget approval before a project can start. Some organizations have a formal process to get a project approved; other organizations have a less-formal process. Although these activities are part of Core Process 1 of the SDLC, they

are often completed well in advance of the start of the first project iteration. They might be called pre-project activities of Core Process 1:

- Identify the problem and document the objective of the solution system.
- Obtain approval to commence the project.

■ System Vision Document

As with all new projects within RMO, a System Vision Document is developed to identify the benefits to the company and the functional capabilities that will be included in the system. Frequently, this is done in two steps: developing a preliminary statement of benefits and then adding estimates of specific dollar costs and dollar benefits. **Figure 1-8** is the System Vision Document for this project.

As described earlier, RMO needs a mobile system that can be used by its purchasing agents as they attend various product, clothing, and fabric trade shows. The system needs to fulfill two major requirements. First, it has to have the functionality to capture information about suppliers and products. Second, it needs to be able to communicate with the home office systems, and because these trade shows are held in various venues around the world, various methods of connectivity are needed.

Preliminary investigation included various equipment options, like notebook computers, tablets, and smartphones. Smartphones appeared to have the best connection options, but their small size made viewing the details of photographs somewhat difficult; the iPad and other portable tablets with advanced connectivity options also appear to be viable options. Because smartphones and tablets have similar requirements, analysts determine to develop an application that will execute on either type of device, giving purchasing agents multiple options for system access.

Toward the end of the initial project activities, a meeting is held involving all the key persons, including a representative of executive management. The decision is made to move ahead with the project and budget the necessary funds.

■ Day 1 Activities

■ RMO—Tradeshow System Overall

The project actually begins with Day 1, which is essentially a planning day. Usually, the project team first reviews the System Vision Document and verifies that the preliminary work is still valid. It reviews the scope of the project to become familiar with the problem, and then it plans the iterations and activities for the remainder of the project. The second SDLC core process—Plan and monitor the project—includes business analysis and project management activities. These Core Process 2 activities are completed on Day 1:

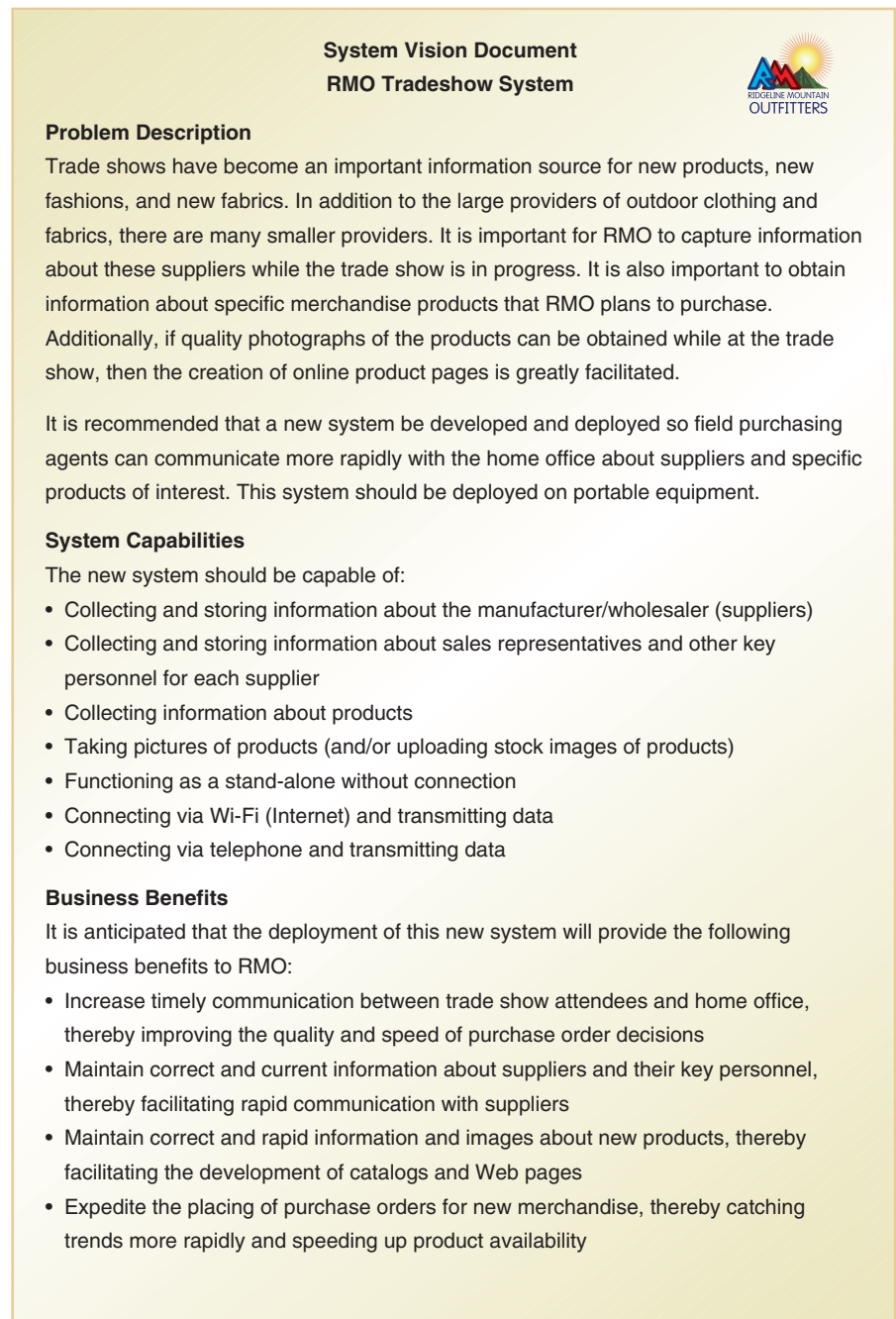
- Determine the major components (functional areas) that are needed.
- Define the iterations and assign each functional area to an iteration.
- Determine team members and responsibilities.

■ Planning the Overall Project and the Project Iterations

Many details need to be considered in a project plan. For our project, we will only focus on the bare essentials. We will describe project planning more elaborately in later chapters. The project team meets with the users to review the overall business need and the objectives of the new system. The System Vision Document serves as the starting point for these discussions. As is often the case, the list of system capabilities provides the foundation information for determining the overall project plan. The first step is to divide the system into several subsystems or components. A **subsystem** is a fully functional part of the complete

subsystem an identifiable and fully functional part of a complete system

FIGURE 1-8 *Tradeshow System Vision Document*



information system. Based on the list of system capabilities, the project team identifies these two subsystems:

- Supplier Information Subsystem
- Product Information Subsystem

The Supplier Information Subsystem will collect and maintain information about the manufacturers or wholesalers and the contact people who work for them. The Product Information Subsystem will capture information about the various products offered by the manufacturers or wholesalers, including detailed descriptions and photographs.

The next step is to identify the order in which the subsystems will be developed. Many issues are considered, such as dependencies between the various tasks, sequential versus parallel development, project team availability, and project urgency. In our case, the team decides that the Supplier Information Subsystem should be scheduled for the first iteration and the Product Information Subsystem should be scheduled for the second iteration. The third and fourth iterations would complete the implementation, testing, and deployment of the system based on what was initially implemented in the first two iterations.

■ Planning the Rest of the First Iteration: The Supplier Subsystem

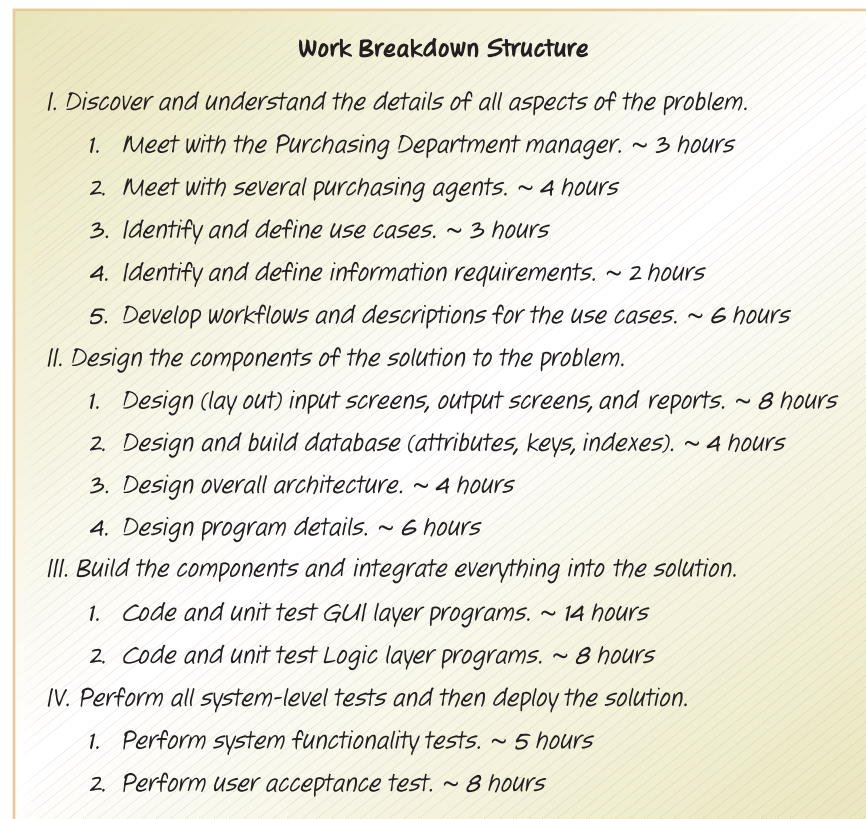
Each iteration is like a system development mini-project. The core processes described earlier can all be applied, with the scope limited to the component that is to be developed during the iteration. The planning process for an iteration consists of these three steps:

- Identify the tasks required for the iteration.
- Organize and sequence these tasks into a schedule.
- Identify required resources (especially people), and assign people to tasks.

The first step is to identify—or attempt to identify—all the individual tasks that need to be done. As these tasks are identified, they are compiled and organized. Sometimes, this organized list of tasks is called a work breakdown structure (WBS). **Figure 1-9** shows the WBS for this iteration.

Part of this effort is trying to estimate how long each task will take. Because this iteration has a very limited scope (and only six days), all the estimates will be in hours. These estimates do not include the time expended by those who are not on the team. However, of those on the team, the estimates include the time for the original work, the time for discussion, and the time for reviewing and checking the WBS for accuracy and correctness.

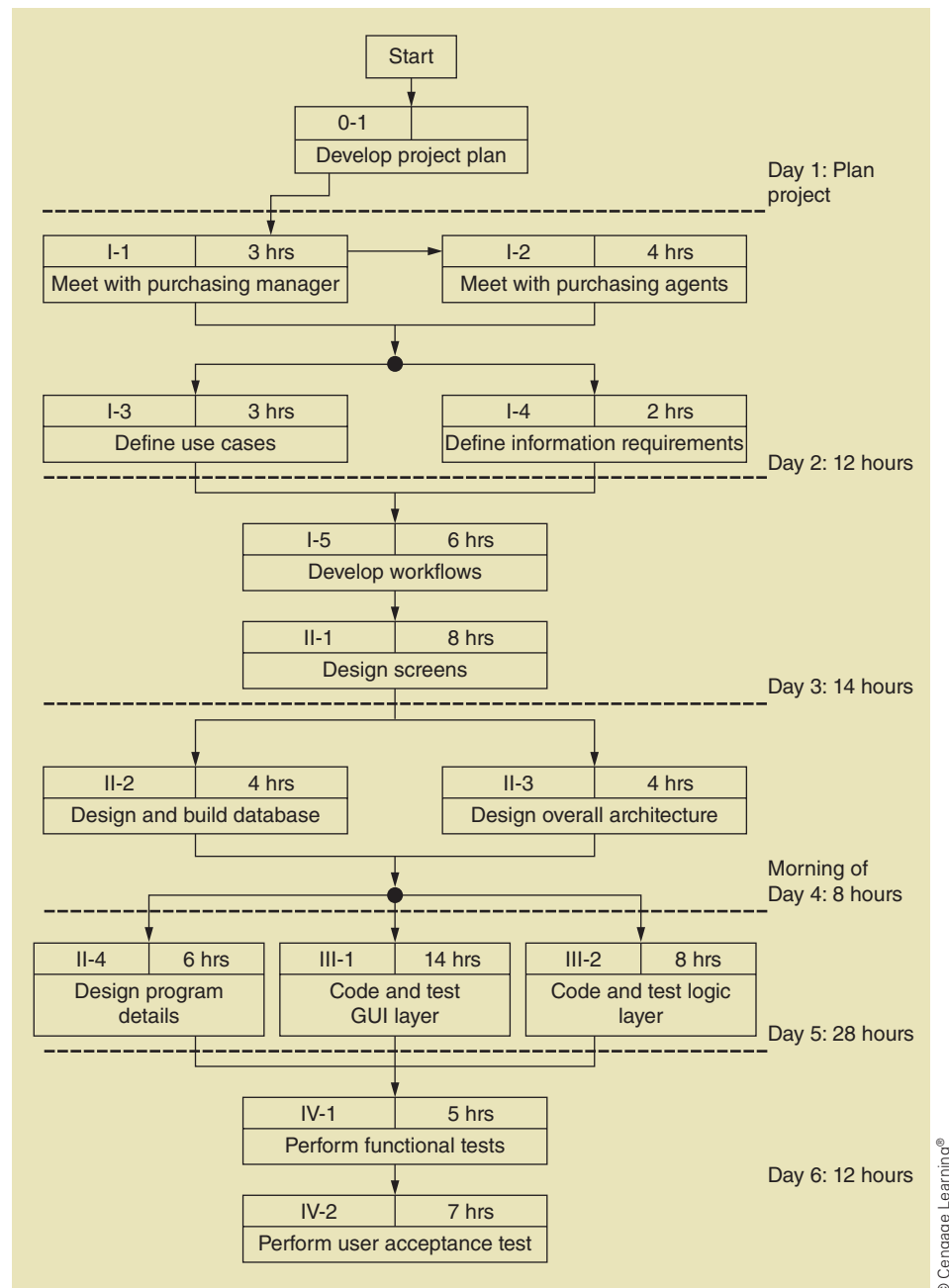
FIGURE 1-9 Sample handwritten work breakdown structure (WBS)



The next step is get these tasks organized into a schedule. Again, we can be very formal and use a sophisticated project scheduling tool, or we can just list the tasks in the order we think they need to be done. Creating the tasks in order is an important part of building the schedule because it identifies any dependencies among the tasks, though many tasks can be done in parallel. For example, it does not make sense to try to design the database before we have identified the information requirements. Again, the great benefit of planning a single iteration is that we can make the schedule informal, and we will be able to adjust the work day-by-day to respond to complexities that occur.

For this iteration, we have taken the tasks from the WBS and placed them on a day-by-day sequence that we call an iteration schedule, as shown in **Figure 1-10**. The project manager can use this diagram to assign people to the tasks and to put the tasks on a specific schedule chart with calendar dates if necessary.

FIGURE 1-10 Schedule for the first iteration



You should be aware that the sequence of activities and the dependencies of those activities are represented in this diagram with only partial accuracy. For example, we show that programming does not start until design has finished. However, in reality, there may be some overlap between the two activities.

The benefit of an iteration schedule is threefold. First, it helps the team organize its work so developers have enough time to think through the critical design issues before programming begins. Second, it provides a measuring rod to see if the iteration is on schedule. For example, if meetings with the purchasing agents take all day or more than a day, the team will know early on that this iteration will take longer than expected. Third, the project manager can see that programming may require more resources if the project is going to stay on this schedule. Hence, the project manager can begin lining up resources to help with that part of the iteration. It should be obvious that even this simple diagram can help a project manager plan and organize the work.

■ Day 2 Activities

■ The Tradeshow System Overall

Day 1 involved planning and organizing the project. Day 2 involves discovering and understanding details, which is a key part of systems analysis. Now we will complete the systems analysis activities in more detail for the complete Tradeshow System. These Core Process 3 activities include:

- Do fact-finding tasks to understand the requirements.
- Develop a list of use cases and a use case diagram.
- Develop a list of classes and a class diagram.

■ Fact Finding and User Involvement

Before the project commenced, a broad definition of requirements was developed. It is now time to examine those requirements and determine exactly what the user needs the system to do. There are various techniques to ensure that the fact finding is complete and thorough. These include interviewing the key users, observing existing work processes, reviewing existing documentation and existing systems, and even researching other companies and other systems. The first step is to identify the key users who will define these details. In this scenario, the manager of the Purchasing Department will be one of the first users to meet with. She will probably designate one or two knowledgeable purchasing agents who can work with the team on an ongoing basis to develop the specifications and to verify that the system performs as required. All successful projects depend on heavy user involvement. In Chapter 2, you will learn more about identifying key stakeholders and gathering information.

■ Identifying Use Cases

Identifying and describing use cases is the way to document what the users need to do with the system, hence, the term *use case*—a case or situation where the system is used. For example, suppose a purchasing agent goes to a trade show and finds a new lightweight sports jacket that will work well for RMO's fall merchandise offerings. Suppose that the first task the purchasing agent needs to do is find out if this supplier has worked with RMO before. Thus, a use case required for the Tradeshow System might be *Look up a supplier*. One good way to help you talk about use cases is to say, "The purchasing agent 'uses' the system to 'Look up a supplier.'" There are multiple methods used to identify use cases, which you will learn later in this book. Some developers prefer to use a similar concept called a *user story*.

FIGURE 1-11 List of use cases for Tradeshow System

Use Case	Description
Look up supplier	Using supplier name, find supplier information and contacts
Enter/update supplier information	Enter (new) or update (existing) supplier information
Look up contact	Using contact name, find contact information
Enter/update contact information	Enter (new) or update (existing) contact information
Look up product information	Using description or supplier name, look up product information
Enter/update product information	Enter (new) or update (existing) product information
Upload product image	Upload images of the merchandise product

© Cengage Learning®

Figure 1-11 is a preliminary list of use cases for the entire Tradeshow System. When the project team meets with the purchasing agents in brainstorming sessions, they identify the use cases together. Because the first iteration is focusing only on the Supplier Information Subsystem, the project team will also focus its attention on only the first four use cases on the list.

■ Identifying Domain Classes

Domain classes identify those things in the real world that the system needs to know about and keep track of. To find domain classes, we look for all objects, or things, that the system uses or captures. Objects come in all types and variations, from tangible items (such as merchandise products that you can see and touch) to more abstract concepts that you cannot touch (such as a promotion), which, though intangible, is an important thing to remember and track. Domain classes are the categories of objects identified, much like a table in a database represents the category of the records it contains. A Product class represents all of the product objects used by the system.

Domain classes are identified during the discussions with purchasing agents by looking for the nouns that describe categories of things. For example, the agents will often talk about suppliers, merchandise products, or inventory items. These nouns become the domain classes, and each domain class has attributes (like contact information, product, or business location) that detail the information you need to store about the domain class.

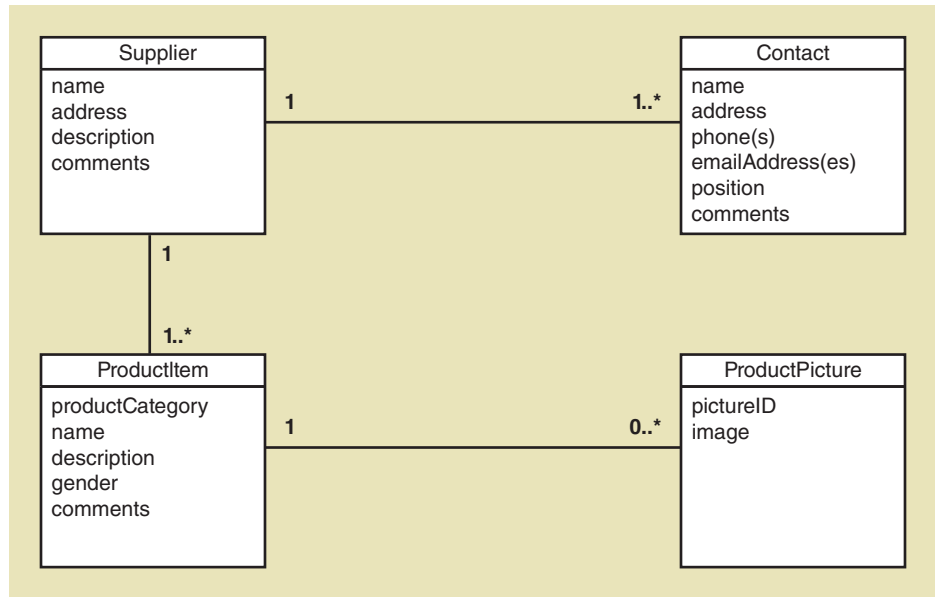
Figure 1-12 illustrates which nouns are fundamental domain classes for the Tradeshow System. The attributes are pieces of information that help define and describe details about a domain class.

FIGURE 1-12 List of domain classes for Tradeshow System

Object Classes	Attributes
Supplier	supplier name, address, description, comments
Contact	name, address, phone(s), e-mail address(es), position, comments
Product	category, name, description, gender, comments
ProductPicture	ID, image

© Cengage Learning®

FIGURE 1-13 Preliminary domain class diagram for the Tradeshow System



In addition to just providing a list of domain classes, systems analysts often develop a visual diagram of the classes, their attributes, and their associations with other classes. This diagram is called a domain class diagram. **Figure 1-13** shows the domain class diagram for the Tradeshow System.

Each box is a class and can be thought of as a particular set of objects that are important to the system. Important attributes of each class are also included in each box. These represent the detailed information about each object that will be maintained by the system. Note that some classes have lines connecting them. These represent associations between the classes that need to be remembered by the system. For example, a contact is a person who works for a particular supplier. A specific example might be that Bill Williams is the contact person for the South Pacific Sportswear Company. Thus, the system needs to associate Bill Williams and the South Pacific Sportswear Company. The association line documents that requirement.

Domain class diagrams are a powerful and frequently used way to understand and document the information requirements of a system. The Tradeshow System is very simple, with only four classes identified—two of which belong to the Supplier Information Subsystem. Most real-life systems are much larger and have dozens or even hundreds of domain classes.

■ Day 3 Activities

The purpose of Day 3 activities is to analyze in detail the use cases and domain classes that are scheduled to be implemented in the first iteration for the Supplier Subsystem. Included are these Core Process 3 activities:

- Perform in-depth fact finding to understand details of each use case.
- Understand and document the detailed workflow of each use case.
- Define the user experience with sketches of screens and reports needed for each use case.

After working with the purchasing agents, developers determined the following use cases pertaining to the Supplier Information Subsystem:

- *Look up supplier*
- *Enter/update supplier information*

- *Look up contact information*
- *Enter/update contact information*

From this list, the developer will then create a use case diagram. A use case diagram is used to graphically portray the use cases and users involved in the subsystem. **Figure 1-14** illustrates a simple use case diagram for the Supplier Information Subsystem showing the four use cases as ovals and the two users as stick figures. The lines connecting users and use cases shows who uses the system for the use case: The purchasing agent carries out all four use cases, but the manager also looks up suppliers or contacts.

The project team will look closely at the steps to follow for each use case to better understand how the application needs to work and to identify what screens and reports will need to be developed. As the team gets more into the details, it may discover that some of the initial analysis is incomplete or not correct and can adjust the WBS to reflect the changes. This is a good time to make such discoveries—much better to find mistakes earlier than after the programs have been written.

■ Developing Use Case Descriptions and Workflow Diagrams

There are various methods for documenting the details of a use case. One that you will learn later in this text is called a *use case description*. Another method is developing an *activity diagram*, which shows all the steps within the use case. The purpose with either method is to document the interactions between the user and the system (i.e., how the user interacts and uses the system to carry out a specific work task for a single use case).

Let us develop an activity diagram for one use case. **Figure 1-15** illustrates the *Look up supplier* use case. The flattened ovals in the diagram represent the activities, the diamonds represent decision points, and the arrows represent the sequence of the flow. The columns designate the person who performs the activities, in this case the purchasing agent in the first column and the Tradeshow System in the second column. Usually, activity diagrams are quite easy for users to understand and critique.

FIGURE 1-14 Use case diagram for the Supplier Information Subsystem

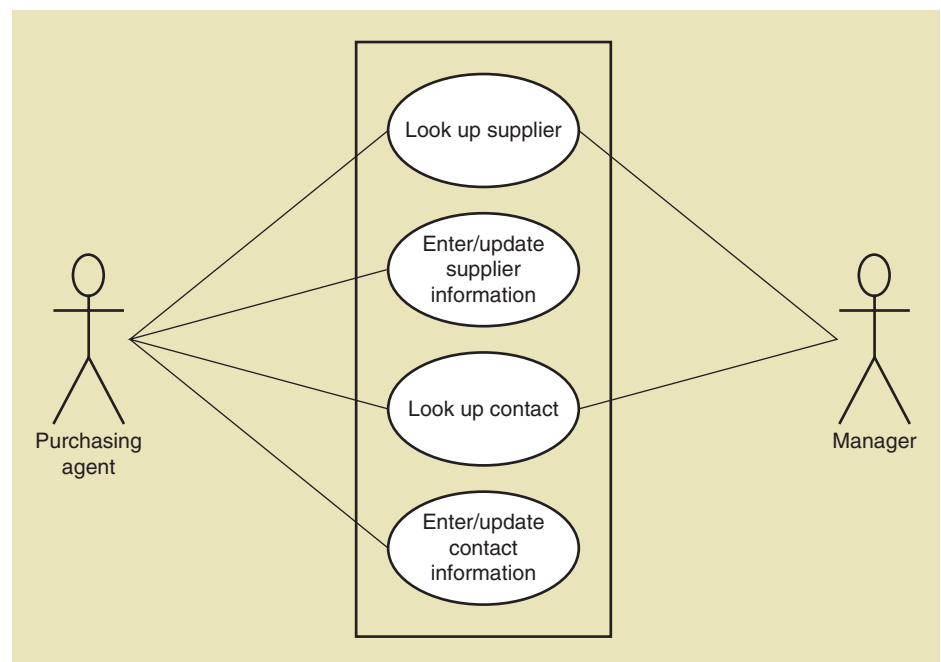
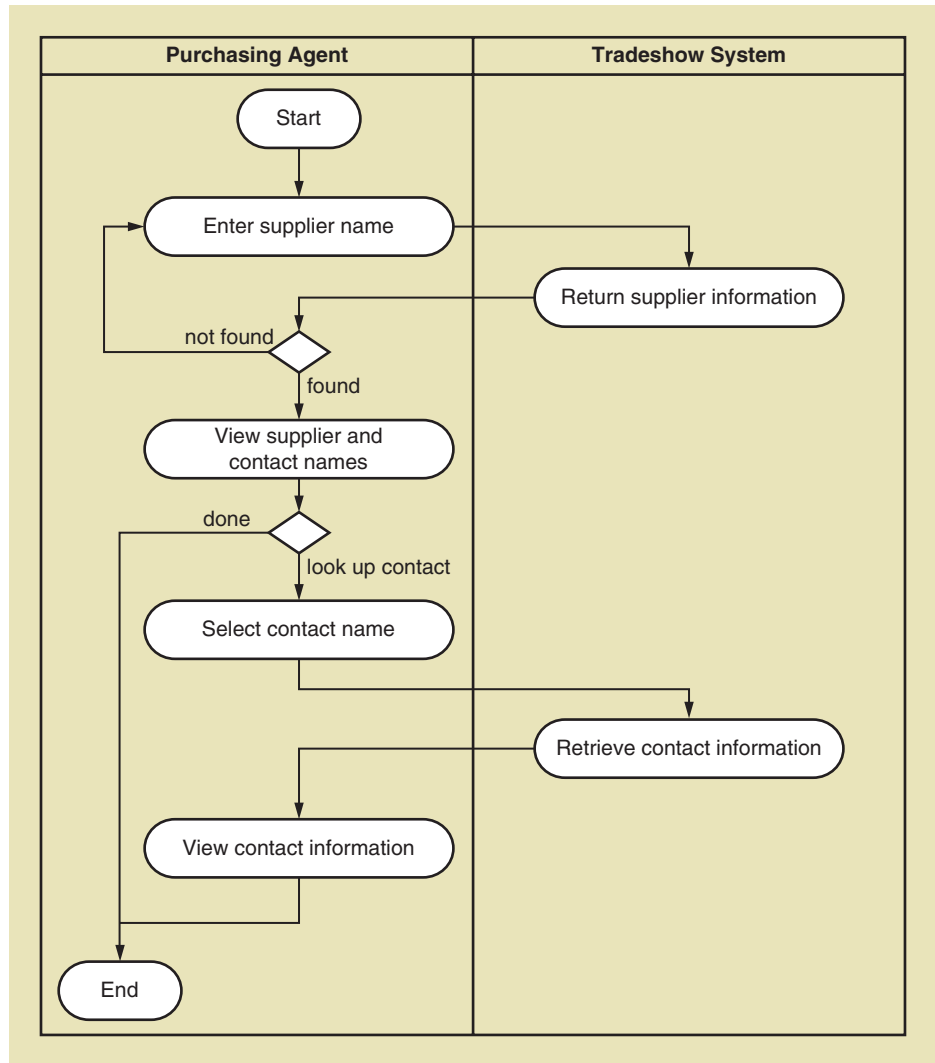


FIGURE 1-15 Activity diagram for the Look up supplier use case



© Cengage Learning®

The arrows that cross the line between users (center) represent the interactions between the user and the system. These are critically important because they designate situations where the developers must provide a screen or Web page that either captures or displays information. These situations become part of the user interface.

In **Figure 1-15**, the top arrow indicates that the supplier name is entered into the system first. Thus, we infer that the use case must have an online lookup page with a field available to enter the supplier name. The next arrow indicates the application requires a page that displays all the details for an individual supplier, including a list of existing contacts. The user may also want to see more details about a specific contact person for this supplier, so the application must provide detailed information request fields for a particular contact. Because the user will need to select one of the displayed results, we must design the page so each entry on the list is either a hot link or can be selected.

■ Defining Screen Layout

User-interface design includes creating how a system looks and how the user interacts with it. Because the user interface is the user's window to the functionality of the system, it is essentially the system itself to the users. If the interface is poorly designed, users will not be able to take full advantage of the system; they may even consider the system to be less than optimal. On the other

hand, a well-designed user interface—one that is intuitive and easy to use, has a full range of features to facilitate navigation, and provides good information—will enhance the utility of the system tremendously.

Figure 1-16 illustrates the layout of the first page used for the workflow in the use case *Look up supplier*. The top half of the page provides the fields where the user enters supplier information, and the bottom portion of the page shows the results. After results are provided, the search box for data entry will remain visible to allow the user to enter another search. Each entry in the results section will be built as a hot link, so the user can click on any particular supplier to retrieve more detailed information. This drill-down technique is a common method used in today's systems and will be intuitively easy for the users.

Data for all of the required fields are stored in the database. Searches seek information in the database, and display the required data, such as name, address, and contact information. An Internet-wide search is also possible. This allows the purchasing agents to look for and view the suppliers' own Web sites, including forums and discussions about the supplier.

■ Day 4 Activities

The primary focus of Day 4 is to design the various components of the solution system, corresponding to Core Process 4: Design System Components. Up to now, we have mostly been gathering the user requirements. On Day 4, we design the system based on the user requirements, which leads to programming efforts. In that sense, design activities can be considered a bridge. The design documents provide the blueprints for how the solution will be structured and how it is to be programmed. System design also tends to involve the technical people, with less need for user participation.

Design can be a complex process. In our small project, we will limit our design examples to only a few models and techniques. Day 4 activities (Core Process 4) include the following:

- Design the database structure (schema).
- Design the system's high-level structure.

FIGURE 1-16 Initial page layout for the Look up supplier use case

Search Results		
Supplier Name	Contact Name	Contact Position

Database design is a fairly straightforward activity that uses the domain class diagram and develops the detailed database schema that can be directly implemented by a database management system. Such elements as table design, key and index identification, attribute types, and other efficiency decisions are made during this activity.

Designing the high-level system structure and the individual programs can be an intricate and complex process. First, the overall structure of the system is designed by identifying the subsystems and connections to other systems. Then, within each subsystem, decisions are made about individual program modules, such as the user interface, business logic, and database access.

It is not uncommon for developers to begin writing program code as they develop portions of the design. However, best practice suggests that the designer complete most of the high-level structure design before writing code. Consequently, in the RMO Tradeshow System project, we will list them as separate activities.

■ Designing the Database

Designing the database uses the information provided by the domain class diagram to determine the tables, the columns in the tables, and other components. Sometimes, the database design is done for the entire system at once or by subsystem. At other times, it is built piecemeal—use case by use case. To keep our project simple, we will just show the database design for the two Supplier Information Subsystem classes. **Figure 1-17** shows the two relational database tables that result with attributes included along with data types and other properties.

■ A General Approach to Design

One of the first questions encountered in software design is how and where to start. So far, we have compiled three sets of information that can answer that question:

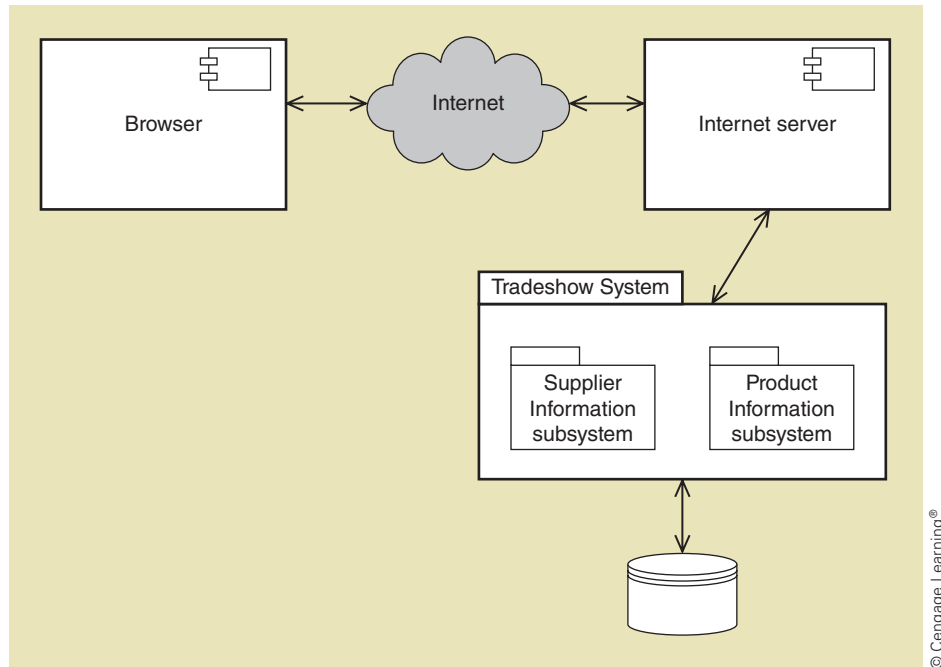
- Use cases, with activity diagrams
- Domain classes, with accompanying diagrams
- Pages and reports, with program and display logic specifications

FIGURE 1-17 Database design for Supplier Information Subsystem

Table Name	Attributes
Supplier	SupplierID: integer {key} Name: string {index} Address1: string Address1: string City: string State-province: string Postal-code: string Country: string SupplierWebURL: string Comments: string
Contact	ContactID: integer {key} SupplierID: integer {foreign key} Name: string {index} Title: string WorkAddress1: string WorkAddress2: string WorkCity: string WorkState: string WorkPostal-code: string WorkCountry: string WorkPhone: string MobilePhone: string EmailAddress1: string EmailAddress2: string Comments: string

© Cengage Learning®

FIGURE 1-18 *Tradeshow System software components diagram*



© Cengage Learning®

Incidentally, in the previous section, we used the class diagram as the basis for the database design. Those same classes are important in developing object-oriented program classes.

Before we jump more into software design, let us briefly discuss the objective of systems design and what we expect the output or result to be. Object-oriented programs are structured as a set of interacting objects based on classes. To program, we need to know what the classes are, what the logic is within each class (i.e., the functions), and which classes must interact. This is the final objective of systems design.

We perform this design by starting at the very highest level and then drilling down to the lowest level until we have defined all the functions within each class. Detailed design documents the thought process of how to program each use case. For Day 4, we focus only on the overall design.

■ Designing the Software Components

Figure 1-18 shows the overall structure of the new system in terms of software components. Although the figure itself appears rather simple, some important decisions have been involved in the development of this design. First, note that the decision was made to build this application as a browser-based system designed for use on smartphones and tablets. A different and very popular approach would have been to build specific smartphone or tablet applications. Browser-based systems sometimes do not provide the same connectivity speed and control as smartphone or tablet applications, but they are more versatile because they are easily deployed on different equipment, such as laptops and tablets, without modification, and on smartphones with only slight modification due to screen size.

These high-level design decisions will determine the detailed structure of the system. A browser-based system is structured and constructed differently than an application system that runs on a smartphone or a tablet computer.

■ Defining the Preliminary Design Class Diagram

The Tradeshow System will be built by using object-oriented programming (OOP) techniques, beginning with developing the set of software classes and

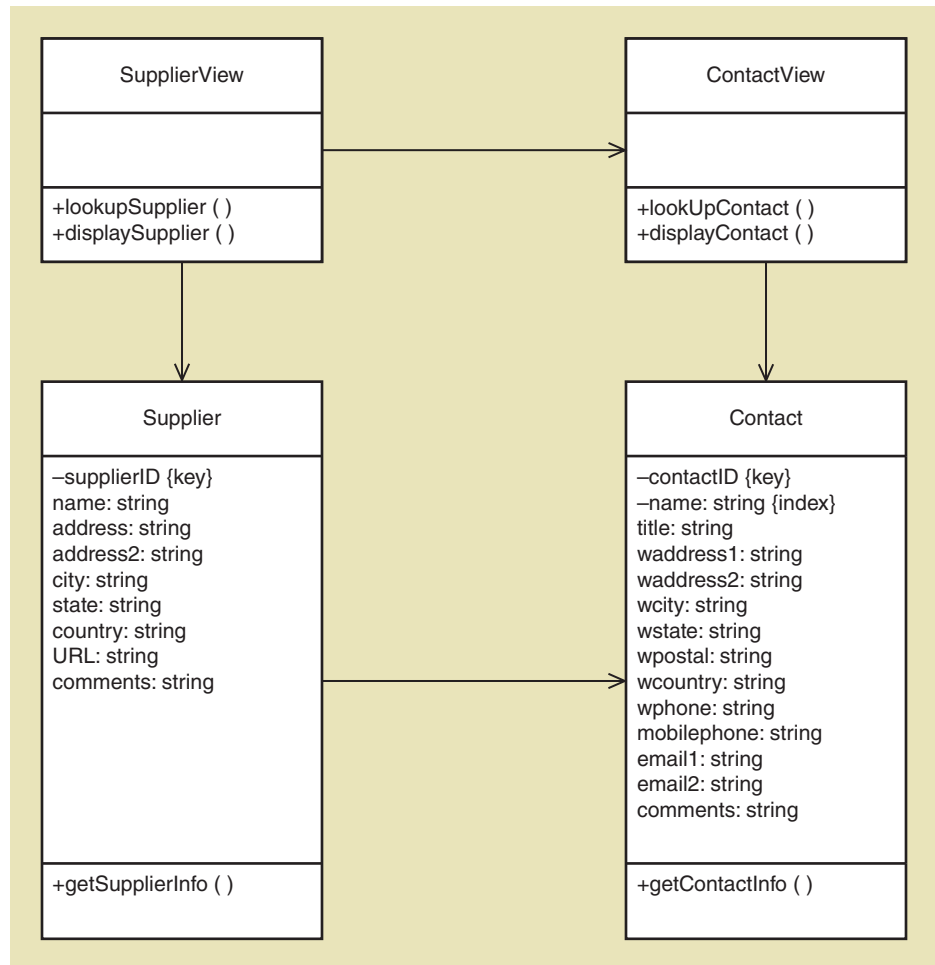
their methods that will be needed for the system. **Figure 1-19** is a preliminary *design class diagram* for the Supplier Information Subsystem that identifies the software classes needed for the system (for example, the SupplierView and ContactView classes). In Figure 1-19, we show only the problem domain design classes and the user-interface View layer classes. Problem domain design classes are usually derived from those classes that were identified during analysis activities—hence, the name: problem (user need) domain design classes. You will also notice that they very closely correspond to the database tables; in fact, in this simple project, they are almost exactly the same as the database tables.

The design classes in Figure 1-19 include the attributes that are needed for the class. They also include method names of the important methods within each class. One final element in the design class diagram is the arrows that show where a class accesses the methods of another class.

■ Designing Subsystem Software Architecture

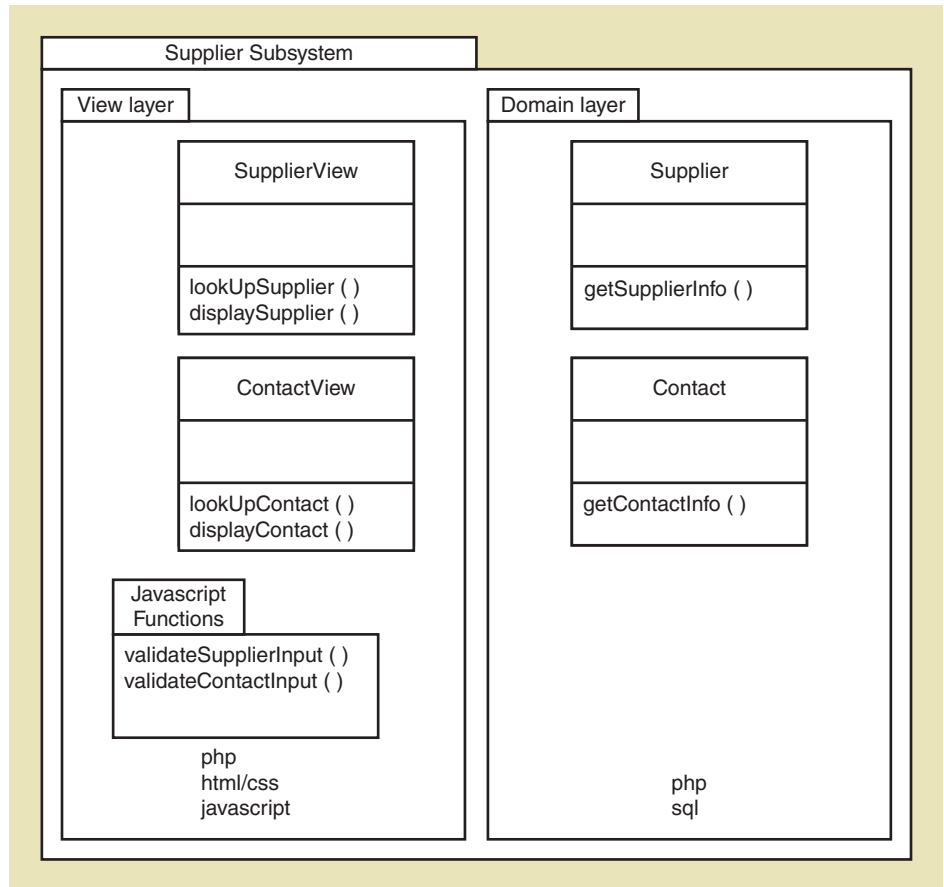
Once we have an overall structure and approach for implementing the new system, we begin to drill down to the software design for the subsystem. **Figure 1-20** illustrates the design of the Supplier Information Subsystem. Notice that this subsystem is further divided into layers: a View layer and a Domain layer. One of the advantages of partitioning the system into layers is it is much easier to build and maintain with this kind of structure given its modular format. For example, the system will be browser based, but different browsers require different techniques. It is better not to get these complexities mixed in with the basic program functions. Hence, they are separated out into a distinct layer.

FIGURE 1-19 Preliminary design class diagram



© Cengage Learning®

FIGURE 1-20 *Supplier Subsystem software architecture*



© Cengage Learning®

The View layer includes two classes that represent what is seen on the user interface—SupplierView and ContactView—as well as some JavaScript functions. The Domain layer includes two classes that interact with each other and with the database—Supplier and Contact.

■ Managing the Project

Design is a complex activity with multiple perspectives—from high-level structural design to low-level detailed program design. In our project, we have separated the tasks for designing the overall system structure from detailed design of the programs themselves. However, these activities are often done concurrently. The basic high-level software architectural structure is defined first, but mid-level and low-level design are often done concurrently with programming.

Detailed design and programming are quite time-consuming activities. A project manager must decide whether to extend the project or bring on additional programmers to help write the code. In our project, we have elected to insert a half-day of free time to bring in two additional programmers and train them. Of course, we could go ahead and begin Day 5's activities to ensure that we keep the project on schedule.

■ Day 5 Activities

As mentioned previously, though detailed design and programming may begin earlier in the project, we identify it as a separate day's activity. We want to emphasize that it is not a good practice to begin programming before critical information is obtained and decisions are made. A much better approach is to understand, design, and build small chunks of the system at a time. Iterative

development anticipates and plans for the expected changes and refinements to the problem requirements that happen during detailed design and programming.

Day 5 activities include the following (Core 5 Process):

- Create detail design.
- Program the subsystem components.

As the programmers write the code, they also perform individual testing on the classes and functions they program. This textbook does not focus on programming activities. However, we include an example of program code so you can see how systems design relates to the final program code. **Figure 1-21** shows PHP code that defines the `SupplierView` class that receives and processes the request for supplier information. Note that it sends a message to the `Supplier` class requesting information when needed.

■ Day 6 Activities

The focus of Day 6 activities is final testing, a step that is required before the system is ready to be deployed. Although there are many types of testing, we mention only two types at this time: overall system functional testing and user acceptance testing. *Functional testing* is usually a test of all user functions and is often done by a quality assurance team. *User acceptance tests* are similar in nature, but these are completed by the users, who test both the correctness of the system and its “fitness” to accomplish the business requirements.

Each of the various testing activities in Day 6 has a similar sequence of tasks to perform. The tasks themselves highly depend on the test data and on the method for testing a particular test case. In some instances, the testing may be automated. In others, individuals may need to manually conduct the tests.

FIGURE 1-21 Code for the `SupplierView` class

```
<?php
class SupplierView
{
    private Supplier $theSupplier;

    function __construct()
    {
        $this->theSupplier = new Supplier();
    }

    function lookupSupplier()
    {
        include('lookupSupplier.inc.html');
    }

    function displaySupplier()
    {
        include('displaySupplierTop.inc.html');
        extract($_REQUEST); // get Form data
        //Call Supplier class to retrieve the data
        $results = $theSupplier->getSupplierInfo($supplier, $category,
                                                $product, $country, $contact);

        foreach ($results as $resultItem) {
            ?>
                <tr>
                    <td style="border:1px solid black">
                        <?php echo $resultItem->supplierName?></td>
                    <td style="border:1px solid black">
                        <?php echo $resultItem->contactName?></td>
                    <td style="border:1px solid black">
                        <?php echo $resultItem->contactPosition?></td>
                </tr>
            <?php }
            include('displaySupplierFoot.inc.html');
        }
    }
?>
```


FIGURE 1-22 Flowchart for testing tasks

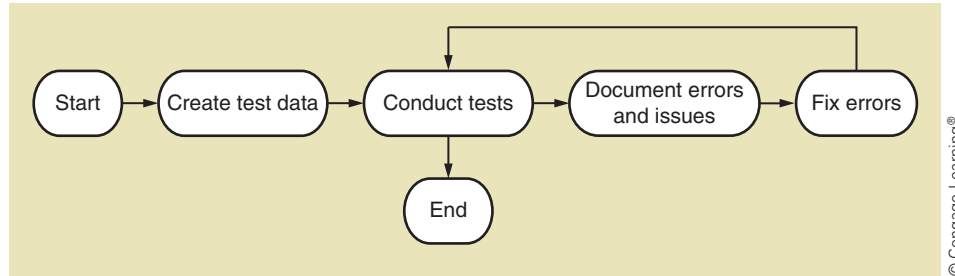


Figure 1-22 is a flowchart for testing the new system. In this flowchart, we have shown the different testing tasks as separate steps; in reality, they tend to all be carried out together. However, any given test case will follow this flow.

■ First Iteration Recap

Figure 1-23 is a screen shot of the browser page that is used in the Tradeshow System to enter and view suppliers. This page will be seen on a tablet. The smartphone page will look different, but it will carry out the same functionality.

As stated previously, this is the first (six-day) iteration of a longer project. Using Agile techniques and iterations within an overall project allows flexibility in defining and building a new system. In this six-day project, the users have had major involvement during all days except Day 4 and Day 5.

A primary problem in developing a new system is that as the project progresses, new requirements are often identified. This happens because the users and the project team learn more about how to solve the business need. Agile, iterative projects are structured to handle these new requirements—often by adding another iteration to the overall project.

As a final step in a current iteration, or perhaps as part of the planning process for the next iteration, there should be a review of the tasks completed and evaluation of the success of the current iteration. The lessons learned and issues to be carried forward create an environment of continual improvement and refinement. Because of this, iterative projects tend to improve and become more efficient during the life of the project.

In this project, the planning for Iteration 2 would confirm the intention of focusing on the use cases and domain classes for the Product Information

FIGURE 1-23 Screen capture for page in Look up supplier use case for tablet

Search Results		
Supplier Name	Contact Name	Contact Position

Subsystem. Efforts would focus on understanding the details of the three use cases and two domain classes required. Activity diagrams or use case descriptions might be created for the complex use cases. Two more database tables would be designed and implemented. Two more software view classes and design domain classes would be defined. Code would then be written and tested to implement the classes and use cases.

■ Where You Are Headed—The Rest of This Book

This seventh edition of *Systems Analysis and Design in a Changing World* includes the printed textbook and supporting online chapters. The current printed textbook provides a compact, streamlined, and focused presentation of those topics that are essential for information system developers. The online chapters extend those concepts and provide a broader presentation of several topics. The online chapters may be integrated into the course or simply used as additional reading as prescribed by the instructor.

We focus on three major subject areas in this book: systems analysis, systems design, and project management. To a lesser extent, we cover systems implementation, testing, and deployment. In addition, we have taken an approach that is quite different from other texts. By providing a basic overview in Chapter 1, we can immediately present in-depth concepts about systems analysis and design in the following chapters. Furthermore, we present project management topics after our discussion on the elements of systems analysis and design, so the reader can use his or her knowledge of necessary analysis and design tasks to understand project management.

■ Part 1: Introduction to System Development

Part 1—comprising Online Chapter A and Chapter 1—presents an overview of systems development. Online Chapter A, “The Role of the Systems Analyst,” describes the many skills required of a systems analyst. It also discusses the various career options available to information systems majors. For those of you who are new to the discipline of information systems, this chapter provides interesting and helpful knowledge about information systems careers. Be sure to find and refer to this online chapter.

This chapter (Chapter 1) provided a detailed, concrete example of what is required in a typical (though simplified) system development project, including processes, techniques, and diagrams. You are not expected to understand all the elements from this brief introduction. However, you should have a general idea of how to approach developing systems. You may want to refer back to this chapter to understand the big picture.

■ Part 2: Systems Analysis Tasks

Chapters 2, 3, 4, and 5 cover systems analysis in detail. Chapter 2 discusses techniques for gathering information about the business problem. Developing the right system solution is possible only if the problem is accurately understood. The people who are affected by the system (the stakeholders) are included in the development of the solution. Chapter 2 explains how to identify and involve the stakeholders, and introduces the concept of models and modeling. Chapters 3 and 4 present methods for capturing the detailed system requirements in a useful format, often models. Chapter 5 presents in-depth models that focus on the details of each use case—use case descriptions, activity diagrams, and system sequence diagrams.

Online Chapter B, “The Traditional Approach to Requirements,” presents a traditional, structured approach to developing systems. To those instructors and

students who desire to learn about data flow diagrams, this chapter provides an in-depth presentation.

■ Part 3: Essentials of Systems Design

Chapters 6, 7, 8, and 9 cover the fundamental concepts related to systems design, including design activities, architectural design, user-interface design, and database design. Chapter 6 provides broad and comprehensive coverage of important concepts and activities of systems design. It serves not only as a broad overview of design principles, but also as a foundation for later chapters that explain the detailed techniques, tasks, skills, and models used to carry out design.

Chapter 7 covers key aspects of architectural design. System interfaces occur when one information system communicates or interacts with another information system without human intervention. System interfaces are becoming increasingly important because of Web services and cloud computing. Other related concepts, such as controls and security, are also presented in this chapter.

Chapter 8 presents design principles related to the user. Designing the user interface requires a combination of analysis and design. It is related to analysis because it requires heavy user involvement and includes specifying user activities and desires. On the other hand, it is a design activity because it specifies final components that are used to drive the programming effort. The pages, forms, reports, and other user interaction components must be precisely designed so they can be programmed as part of the final system.

Chapter 9 explains how to design the database from the information gleaned during analysis and the identification of the domain classes.

■ Part 4: Projects and Project Management

By this point, you will have a basic understanding of all the elements of system development.

Part 4 uses these concepts to explain the process of organizing and managing development projects. Chapter 10 describes different approaches to system development in today's environment, including important system development life cycle models. It is an important chapter to help you understand how projects actually get executed. Additionally, Agile development is discussed in more detail along with two specific Agile system development methodologies—Agile Unified Process and Scrum.

Chapter 11 teaches foundational principles of project management. Every systems analyst helps organize, coordinate, and manage software development projects. In addition, many analysts will become team leaders and project managers. The principles presented in Chapter 11 are essential to a successful career and they are related specifically to Agile methodologies.

Online Chapter C, “Project Management Techniques,” goes into more detail regarding the tools and techniques used by systems analysts and project managers to plan and monitor development projects. For those instructors and students who would like to learn advanced project management skills, this is an important chapter.

■ Part 5: Advanced Design and Deployment Concepts

Chapters 12 and 13 explain the models, skills, and techniques used to design software for each use case, including designing for multilayer software. The objective of these two chapters is to teach you the techniques—from simple to complex—that can be used to effectively design software. Chapter 14 describes the final elements in system development: testing, deployment, maintenance, and version control.

CHAPTER SUMMARY

This chapter provided an overview of an information systems development project called the Tradeshow System. An information system is a set of interrelated components that collect, process, store, and provide as output the information needed to complete business tasks. Systems analysis is the set of activities used to understand and document what the new information system should accomplish. Systems design involves describing in detail the resulting system.

The system development life cycle (SDLC) identifies all of the activities required to research, build, launch, and maintain an information system. Most SDLCs include the following six core processes, although names used for the processes vary:

1. Identify the problem and obtain project approval.
2. Plan and monitor the project.
3. Discover and understand the details of the problem.
4. Design the system components that solve the problem.
5. Build, test, and integrate system components.
6. Complete system tests and then deploy the solution.

A system development methodology (also called a development process) provides a comprehensive set of guidelines for carrying out the core processes and corresponding activities and tasks for the SDLC. Agile development refers to recent methodologies that emphasize flexibility and rapid response to change. Iterative development is an approach to the SDLC where the system is “grown” through a series of mini-projects. Agile development is highly iterative.

The activities and tasks that support the six core processes of the SDLC were explained as we went through an implementation of one subsystem of the Tradeshow System. We divided the project into initial project activities that comprised six project work days to show how the first iteration of the Tradeshow System was completed.

KEY TERMS

Agile development
computer application (app)
information system
iterative development

project
subsystem
system development life cycle (SDLC)

system development process (methodology)
systems analysis
systems design

REVIEW QUESTIONS

1. What is the difference between an information system and a computer application?
2. What is the purpose of systems analysis? Why is it important?
3. What is the difference between systems analysis and systems design?
4. What is a project?
5. What is the purpose of the system development life cycle (SDLC)?
6. What are the six core processes of the SDLC?
7. What is meant by Agile development and iterative development?
8. What is the purpose of a System Vision Document?
9. What is the difference between a system and a subsystem?
10. What is the purpose of a work breakdown structure (WBS)?
11. What are the components of a work breakdown structure (WBS)? What does it show?
12. What information is provided by use cases and a use case diagram?
13. What information is provided by a domain class diagram?
14. How do a use case diagram and a domain class diagram drive the system development process?
15. What is an activity diagram? What does it show?
16. How does an activity diagram help in user-interface design?
17. What is the purpose of software component design?
18. What new information is provided in a design class diagram (more than a domain class diagram)?
19. What are the steps of system testing?
20. What is the purpose of user acceptance testing?
21. Why is it a good practice to divide a project into separate iterations?
22. What should be the primary objective of each iteration?

PROBLEM AND EXERCISES

1. Consider the example that described the responsibilities of a property owner, an architect, and a contractor when creating a new building. Create a similar analogy for a high school reunion project where there is the reunion committee, a professional event planner, and a hotel event vendor that would manage the actual event.
2. The chapter described several information systems (or subsystems) that are used to support students. For your university or college, list at least four information systems (or subsystems) that are used to support the work of faculty and staff.
3. For each of the following chapter figures, which ones show the Tradeshow System overall and which figures show the components that apply to the Supplier Subsystem?
 - a. System Vision Document (Figure 1-8)
 - b. Work Breakdown Structure (Figure 1-9)
 - c. List of Use Cases (Figure 1-11)
 - d. List of Domain Classes (Figure 1-12)
 - e. Domain Class Diagram (Figure 1-13)
 - f. Use Case Diagram (Figure 1-14)
 - g. Activity Diagram (Figure 1-15)
 - h. Initial Screen Layout (Figure 1-16)
 - i. Database Design (Figure 1-17)
 - j. Component Diagram (Figure 1-18)
 - k. Design Class Diagram (Figure 1-19)
 - l. Software Architecture (Figure 1-20)
 - m. Code for SupplierView Class (Figure 1-21)
 - n. Screen Capture (Figure 1-23)
4. For the same chapter figures above, which are for planning, for analysis, for design, for implementation, or for testing?
 - a. System Vision Document (Figure 1-8)
 - b. Work Breakdown Structure (Figure 1-9)
 - c. List of Use Cases (Figure 1-11)
 - d. List of Domain Classes (Figure 1-12)
 - e. Domain Class Diagram (Figure 1-13)
 - f. Use Case Diagram (Figure 1-14)
 - g. Activity Diagram (Figure 1-15)
 - h. Initial Screen Layout (Figure 1-16)
 - i. Database Design (Figure 1-17)
 - j. Component Diagram (Figure 1-18)
 - k. Design Class Diagram (Figure 1-19)
 - l. Software Architecture (Figure 1-20)
 - m. Code for SupplierView Class (Figure 1-21)
 - n. Screen Capture (Figure 1-23)

CHAPTER CASE

Keeping Track of Your Geocaching Outings

When Wayne Johansen turned 16, his dad bought him a new Garmin handheld GPS system. His family had always enjoyed camping and hiking, and Wayne was usually the one who monitored their hikes with his dad's GPS system. He always liked to carry the GPS to monitor the routes, distances, and altitudes of their hikes. More recently, though, he had found a new hobby using his GPS system: geocaching.

Geocaching is a high-tech version of the treasure hunts that most of us did when we were kids. Participants search for geocaches or caches that are small, hidden, waterproof containers that typically contain a logbook and perhaps a small item. When found, the participant sometimes gets instructions for the next move—to either enter information into a logbook or to look for the next cache.

As Wayne became more involved with his hobby, he discovered that there are many different kinds of activities for geocaching enthusiasts. The simplest ones

are those that involve caches found by using GPS coordinates, although even some of these tasks can be difficult if the caches are well hidden. Some of the activities involve multipoint drops where there is a set of clues in multiple locations that must be followed to arrive at the final cache point. Some activities involve puzzles that must be solved to determine the coordinates of the final cache.

Before long, Wayne wanted to make his own caches and post them for people to find. He discovered that there were several Web sites with access to geocaching information, caches, and memberships. He joined one of the geocaching Web sites and used it to log his finds. But he decided he would like to create his own system for tracking all the information he had about his caches. Conveniently, Wayne's older brother Nick, a college student majoring in information systems, was looking for a semester project for one of his programming classes. The two of them decided to

develop a system to help Wayne keep track of all his geocaching activities.

In this end-of-chapter case, you will go through the various core processes of an SDLC and perform some of the activities of a development project. The project is divided into days, as was our Tradeshow System project. The daily assignments for this case should be considered as preliminary efforts and rough drafts. The objective of these assignments is to help you remember the overall approach to software development. Several assignments have been listed for each day to allow your instructor to select those that best meet the objectives of the course.

Day 0: Define the Vision

Either by yourself or with another class member, brainstorm all the functions this geocaching system might do. Keep it at a very high level. These activities closely relate to Core Process 1: Identify problem and obtain approval.

Assignment D0-1: Write a rough draft of the System Vision Document based on your brainstorming ideas. [Hint: Think of what Wayne wants the system to do and why this helps him.]

Day 1: Plan the Project

Based on the scope and vision you described in the System Vision Document, divide the project into at least two separate subsystems that can be done in separate iterations. For example, perhaps a first version can run on a laptop, with a second version that includes mobile components for a smartphone. These activities are related to Core Process 2: Plan and monitor the project—what to do, how to do it, and who does it.

Assignment D1-1: Divide the system into at least two separate components or subsystems, which can be supported with two iterations. Briefly describe each.

Assignment D1-2: Create a work breakdown structure that lists all the steps to complete the first iteration. Put a time estimate on each step. [Hint: Use the one in this chapter as a model.]

Day 2: Define and Understand the Requirements

On Day 2, you want to get an overall view of what the system needs to do for Wayne. There are two primary areas to focus on to obtain this high-level understanding of the system: a list of use cases and a list

of domain classes. You could document this information in lists, but diagrams provide a visual representation that is often easier to remember and understand. These activities support Core Process 3: Discover and understand details.

Assignment D2-1: Identify a few use cases that apply to one subsystem. [Hint: Think of what Wayne plans to do with the system. He will use the system to “do what”?]

Assignment D2-2: Try to identify the classes that apply to the first project iteration. [Hint: Think of “information things” that Wayne wants the system to “remember.”]

Assignment D2-3: Create a simple use case diagram from the list of use cases. [Hint: Drawing by hand is fine. Use the one in this chapter as a model.]

Assignment D2-4: Create a simple class diagram from the list of classes. [Hint: Drawing by hand is fine. Use the one in this chapter as a model. Think of other information that applies to each class.]

Day 3: Define the User Experience

These activities are a continuation of what you began in Day 2. The objective here is to further define what Wayne will need and how he will actually use the system. You will determine exactly how each use case works—what steps and options are available with the use case and even what the display and data-entry screens will look like. These activities primarily support Core Process 3: Discover and understand details.

Assignment D3-1: Select a single use case and identify the steps required to perform the use case. [Hint: Think of what Wayne does and how the system responds.]

Assignment D3-2: Make a workflow diagram of the selected use case. [Hint: Drawing by hand is fine. Each step from D3-1 goes in an oval. Connect the ovals with arrows.]

Assignment D3-3: Sketch out one of the screens that will be required to support a use case. The screen should allow for data entry and display of information. [Hint: Don’t make it elaborate. Focus only on the input and output data fields that apply to only one use case.]

Day 4: Develop the Software Architecture Design

The high-level software architectural design of the system generally includes decisions about how the system will be built and what the database will look like. Design is a technical activity that requires experience in programming, database development, and software architecture. These activities support Core Process 4: Design system components.

Assignment D4-1: Design a preliminary database schema for the classes in this iteration. [Hint: Each class becomes a table. The attributes become table columns.]

Assignment D4-2: Decide whether you will build a desktop system or a browser-based system. Write a couple of paragraphs listing the pros and cons of each alternative to defend your decision. [Hint: Either option is valid. Think of reasons to support your decision.]

Day 5: Develop the Detailed Design and Program the System

You probably have had many class projects where you designed a system and then programmed it. These kinds of activities support Core Process 5: Build, test, and integrate system components.

Assignment D5-1: Write a paragraph describing what programming language(s) you would recommend and what development environment

you prefer. For this answer, draw on your previous programming and development experiences. [Hint: There are many valid solutions. Give reasons for your preference.]

Day 6: Test and Deploy the System

You may have had opportunities to perform comprehensive testing of your programming class projects, especially if you have developed systems that integrated with other systems. These activities support Core Process 6: Complete system tests and deploy solution. Obviously, you can only do this if you have programmed the system.

Assignment D6-1: Write a paragraph describing the difference between programmer testing and user testing. [Hint: Why is it hard to test your own work? What do the users know that you don't know?]

Assignment D6-2: Write a paragraph describing all the issues that might need to be addressed to deploy this system. [Hint: You might want to search the Internet to learn about deployment issues.]

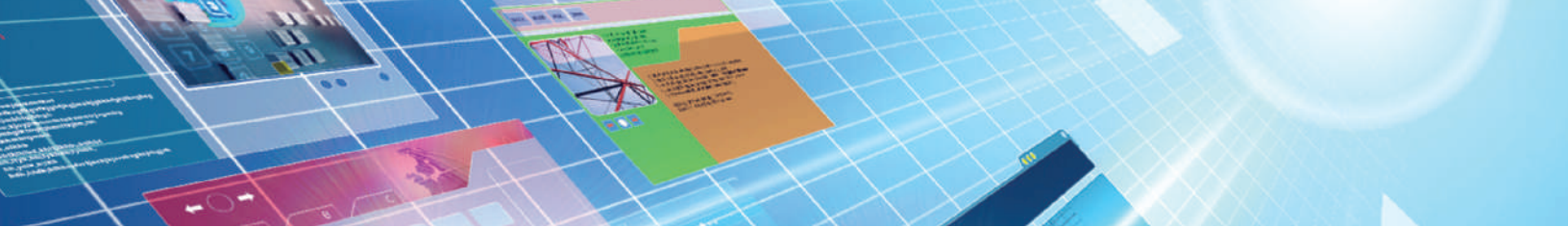
Assignment D6-3: Look at www.geocaching.com, which is a commercial Web site. What other issues need to be addressed to deploy this type of Web site? [Hint: Think about all the issues related to security, robustness, financial protection, high volumes, uptime, different browsers, and so forth.]



Systems Analysis Activities

PART TWO

- ▶ **Chapter 2**
Investigating System Requirements
- ▶ **Chapter 3**
Identifying User Stories and Use Cases
- ▶ **Chapter 4**
Domain Modeling
- ▶ **Chapter 5**
Use Case Modeling
- ▶ **Online Chapter B**
The Traditional Approach to Requirements



CHAPTER TWO

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- › Describe the activities of systems analysis
- › Explain the difference between functional and nonfunctional requirements
- › Identify and understand different kinds of stakeholders and their contributions to requirements definition
- › Describe information-gathering techniques and determine when each is best applied
- › Describe the role of models and UML in systems analysis
- › Develop UML activity diagrams to model workflows

CHAPTER OUTLINE

- › The RMO Consolidated Sales and Marketing System Project
- › Systems Analysis Activities
- › What Are Requirements?
- › Stakeholders
- › Information-Gathering Techniques
- › Models and Modeling
- › Documenting Workflows with Activity Diagrams

OPENING CASE MOUNTAIN VISTA MOTORCYCLES

Amanda Lamy, president and majority stockholder of Mountain Vista Motorcycles (MVM), is an avid motorcycle enthusiast and businesswoman. MVM is headquartered in Denver and has locations throughout the western United States and Canada. Since the late 1990s, the market for motorcycles has grown tremendously. Amanda expects that the market will continue to be strong throughout the 2010s.

The demographics of the motorcycle market are an interesting study in contrasts—an over-50 majority customer base, but with an under-30 contingent that is growing. The remaining customer base includes men and women between the ages of 30 and 50.

The “over 50” customers are typically male professionals or businesspeople who are partly or fully retired. They have substantial disposable income, lots of free time, and tend to own multiple expensive motorcycles from such manufacturers as Harley-Davidson, Honda, Ducati, and Moto Guzi. Older customers are generally comfortable with Internet and Web technology, but are not significant users of social media technology. Although many own smartphones, they tend to use them primarily for voice, e-mail, and texting.

Male customers under 30 years of age tend to buy sport and dirt bikes, typically from such manufacturers as Suzuki and Kawasaki. They buy less-expensive bikes than older customers and are more likely to buy parts and supplies from MVM to service their own bikes. Female customers under 30 years of age tend to buy motor scooters and smaller “commuter” motorcycles. Customers in the 30–50 age range include men and women who buy bikes of many types from many manufacturers. For all customers under the age of 50, comfort with and use of Internet technology, social media, and portable computing devices such as smartphones and iPads is very high, but especially with customers under age 30.

Amanda is convinced that the key to long-term success in the motorcycle market is to build an active community of motorcycle enthusiasts that includes a broad spectrum of customers at each MVM location. In essence, each location needs to be seen as a hub of local motorcycle-related activity and information in physical and virtual terms. On the physical side, MVM has added activity and event-oriented pages to its Web sites, sponsored rallies and clubs, added meeting rooms and small coffee shops in some locations, and collocated with bars and restaurants that feature motorcycle-related themes and entertainment. These efforts have yielded good results with older customers but less so with younger customers.

Amanda is concerned about the lack of participation in the new initiatives by younger customers and is sure that MVM’s minimal presence in social media and virtual relationships is a significant factor. She and her senior staff, most of whom are older, are unsure how to attract younger customers. They have little knowledge of and no experience with creating modern technology-based virtual communities. To fill this gap, Amanda turns to her chief information officer (CIO), and tasks her with finding a way technology can connect the diverse customer base.

MVM’s CIO is starting to develop a project plan for a virtual community oriented toward younger customers. If the plan were for developing a traditional information system, she would use such standard approaches as interviewing internal users and managers, and having her development staff write specifications, generate storyboards and screen layouts, and develop prototypes. But few of the intended virtual community users are MVM employees, and few staff members fully comprehend how to successfully use social media and other techniques for building virtual societies. Traditional methods of defining and refining requirements seem inadequate to the task.

■ Overview

In Chapter 1, you saw the system development life cycle (SDLC) being employed by Ridgeline Mountain Outfitters (RMO) for a small information system application called the Tradeshow System. Development of that system followed the six core processes of the SDLC. The information system application envisioned for Mountain Vista Motorcycles is another situation where the six core processes of the SDLC would be required, but because there is much uncertainty about what the virtual community would actually involve, the project would have to be highly agile. Some of the early iterations would need to focus intensely on discovering and understanding the details of the problem and less on planning the project up front. In this respect, each project adapts the SDLC to its specific needs. Nevertheless, the same six core processes are always required:

1. Identify the problem or need and obtain approval to proceed.
2. Plan and monitor the project—what to do, how to do it, and who does it.

3. Discover and understand the details of the problem or the need.
4. Design the system components that solve the problem or satisfy the need.
5. Build, test, and integrate system components.
6. Complete system tests and then deploy the solution.

In this chapter, we start expanding the scope and detail of the SDLC core processes to cover a wider range of concepts, tools, and techniques. The extra depth and detail are needed to tackle larger and more complex projects. This chapter concentrates on systems analysis activities (Core Process 3), and the next few chapters follow up with detailed discussions of models developed during those systems analysis activities. Subsequent chapters expand the discussion of other core SDLC processes. The RMO Consolidated Sales and Marketing System project is used for examples throughout the rest of the book. It is a much larger project than the Tradeshow System project in Chapter 1.

■ The RMO Consolidated Sales and Marketing System Project

Ridgeline Mountain Outfitters has an elaborate set of information systems applications developed over the years to support operations and management. However, there is a growing gap between customer expectations, current technological capabilities, and existing RMO systems that support sales and customer interaction. This section reviews the existing system inventory and introduces the proposed Consolidated Sales and Marketing System that will update and enhance sales and marketing.

■ Existing RMO Information Systems and Architecture

RMO's Information Systems Department has always been forward looking. In past years, the department, in conjunction with corporate strategic plans, has developed five-year plans for development and deployment of new technology and information systems. The planning process has been an excellent tool to help the organization stay current with new trends and technology capabilities. However, the plans themselves have had mixed success. One of the problems with long-range information technology (IT) and software development plans is that technology changes rapidly and moves in unexpected directions. For example, the Tradeshow System described in Chapter 1 was made possible by the availability of powerful and flexible handheld devices and the widespread availability of Wi-Fi and Internet connections. Both technologies reached a fairly mature level in just a couple of years, which created an opportunity for RMO to optimize this important business process for mobile technology.

Historically, RMO has adopted new technology as soon as it became cost effective. Past examples include adoption of smaller servers and desktop computing, interconnection of locations with dedicated telecommunications links, and such Web-based technologies as customer-oriented Web sites and browser-based user interfaces for internal systems. At present, RMO has a disparate collection of computers dispersed across home offices, retail stores, telephone centers, order fulfillment/shipping centers, and warehouses—everything connected by a complex set of local area networks (LANs), wide area networks (WANs), and virtual private networks (VPNs). This constitutes RMO's current technology architecture.

The term **technology architecture** refers to the set of computing hardware, network hardware and topology, and system software—such as operating and database management systems—employed by an organization. RMO's technology architecture is modern but not state of the art, which is consistent with its goal of adopting proven technology.

technology architecture a set of computing hardware, network hardware and topology, and system software employed by an organization

application architecture the set of information systems (the software applications) the organization needs to support its strategic plan

The term **application architecture** refers to the set of information systems (the software applications) the organization needs to support its strategic plan. Each information system supports the work that needs to be carried out by the organization; the strategic plan includes what the information systems are and how the information systems are integrated together.

Currently, the major systems in RMO's application architecture consist of the following:

- **Supply chain management (SCM).** This application was deployed five years ago as a client/server application using Java and Oracle. Currently, it supports inventory control, purchasing, and distribution, although integration of functions needs improvement. The new Tradeshow System will interface with this system.
- **Phone/mail order system.** This modest client/server application was developed 12 years ago using Visual Studio and Microsoft SQL Server as a quick solution to customer demand for phone and mail orders. It is integrated with the SCM and has reached capacity.
- **Retail Store System (RSS).** This is a retail store package with point-of-sale processing. It was upgraded eight years ago from overnight batch to real-time inventory updates to/from the SCM.
- **Customer Support System (CSS).** This system was first deployed 15 years ago as a Web-based catalog to support customer mail and phone orders. Four years later, it was upgraded to an Internet storefront, supporting customer inquiries, shopping cart, order tracking, shipping, back orders, and returns.

All organizations—including RMO—face a difficult challenge keeping all their information systems current and effective. Because development resources are limited, an organization's technology architecture and application architecture will include a mix of old and new. Older systems were often designed for outdated operational methods and typically lack modern technologies and features that some competitors have adopted to improve efficiency or competitiveness. Such is the case with RMO's existing customer-facing systems, which have several shortcomings, including:

- Treating phone, Web, and retail sales as separate systems rather than as an integrated whole
- Employing outdated Web-based storefront technology
- Not supporting modern technologies and customer interaction modes, including mobile computing devices and social networking

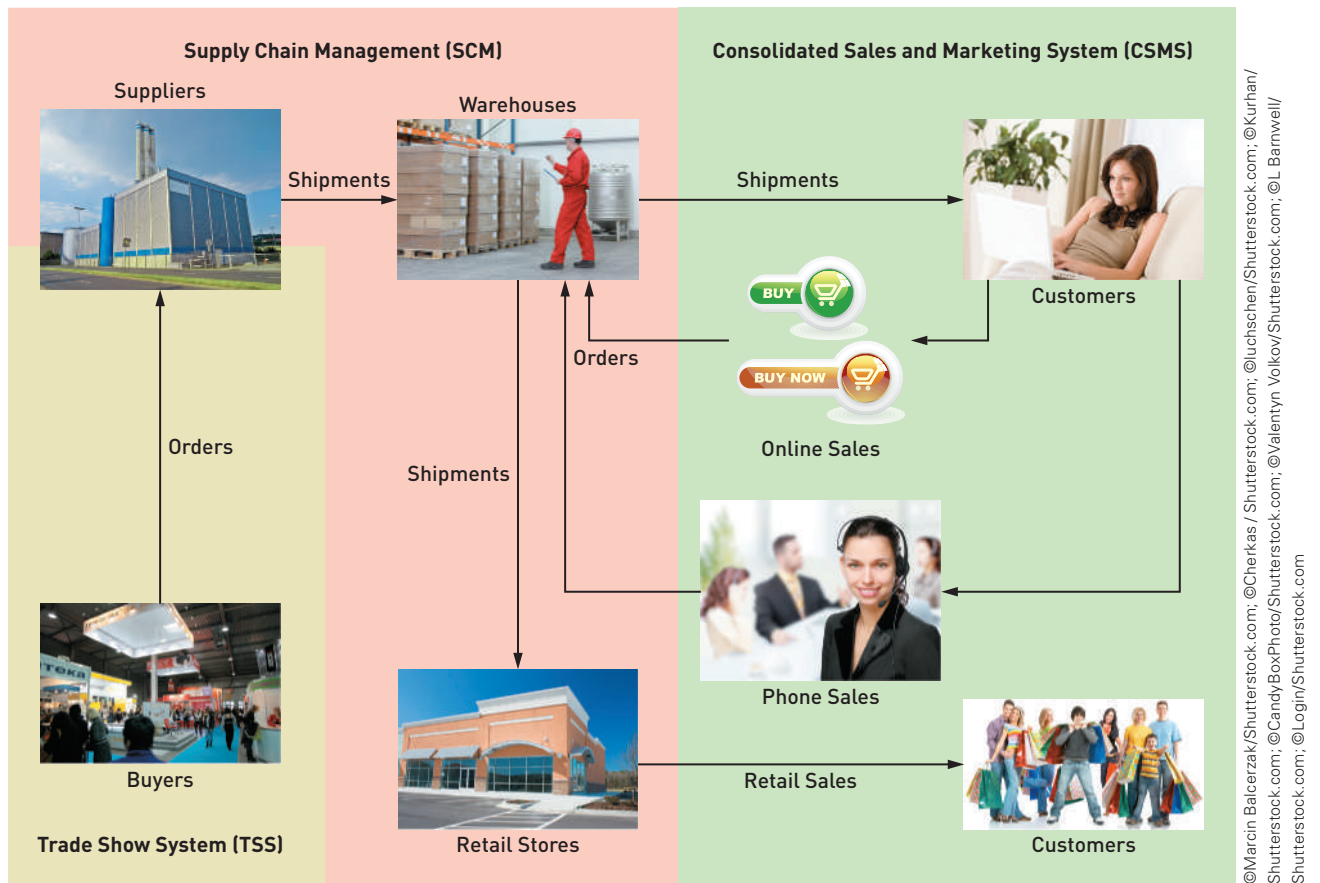
Rather than incrementally update the existing sales systems, RMO plans to replace them with the new Consolidated Sales and Marketing System, as shown in **Figure 2-1**.

■ The New Consolidated Sales and Marketing System

The goals of the Consolidated Sales and Marketing System (CSMS) are to modernize the technology and functionality of the CSS and to add more customer-oriented functionality. On the technology side, the CSMS will incorporate current Web standards and require high-bandwidth customer Internet connections and high-resolution displays. Updating the technology allows a higher degree of interactivity, richer graphics, and a streamlined interface.

Key additions to system functionality will be support for mobile computing devices, incorporation of customer feedback and comments into product information, and integration of social networking functions. Unlike the CSS, the CSMS will support smartphones and tablet computers with interfaces specifically designed for each platform and with downloadable apps. Customer feedback will be captured directly through the Internet storefront, from

FIGURE 2-1 Proposed application architecture for RMO (partial)



RMO-supported comment forums and blogs, and mined from Facebook and Twitter. RMO will develop a complete presence in each social networking venue and enable system users to share purchases, recommendations, coupons, and store credits using those venues.

The new CSMS will have four subsystems:

- The Sales subsystem provides such basic functions as searching the online catalog, purchasing items, and paying for them online. However, it has many new capabilities to assist the shopper. The system will provide specific suggestions about accessories that go with the purchased item. Images and videos of animated models will be available to help the customer see how various items and accessory packages will look together. The system will also provide information to shoppers about related purchases made by other shoppers. Customer ratings and comments are available for viewing. Finally, key social networking components will permit shoppers to network with their friends by sending messages to ask their opinions about particular merchandise items.
- The Order Fulfillment subsystem will perform all the normal tasks of shipping items and allowing customers to track the status of their orders and shipments. In addition, as part of order fulfillment, customers can rate and make comments about particular merchandise and their overall shopping experience. They may also make suggestions directly to RMO about the attractiveness of the Web site and the quality of the service they received.
- The Customer Account subsystem provides services that enhance the customer experience. Customers can view and maintain their account information. They also can “link up” with friends who are also customers to share

experiences and opinions on merchandise. The system will keep track of detailed shipping addresses, including payment information and preferences. RMO also instituted a Mountain Bucks program where customers accumulate credits that can be used to redeem prizes as well as purchase merchandise. Customers may use these Mountain Bucks for themselves or they may transfer them to other people in their family/friends group. This is a great opportunity to combine accumulated bucks to obtain expensive merchandise.

- The Marketing subsystem is primarily for employees to set up the information and services for customers. Additionally, employees can enter information about the merchandise offered by RMO. This subsystem is also fed by the SCM system to maintain accurate data on the inventory in stock and anticipated arrival dates of items on order. Employees also set up the various promotional packages and seasonal catalogs by using the functions of this subsystem. Furthermore, RMO is experimenting with a new idea to enhance customer experience and satisfaction: building partner relationships with other retailers so customers can earn “combined” points with RMO purchases or a partner retailer purchase. These points can be used at RMO or transferred and used at the partner. For example, because RMO sells outdoor and sporting clothing, it has partnered with various sporting goods stores. That way, a person can buy sporting equipment at the sporting equipment store and get promotional discounts for clothing at RMO. The success of this new venture has yet to be proven, but RMO anticipates that it will enhance the shopping experience of all its customers.

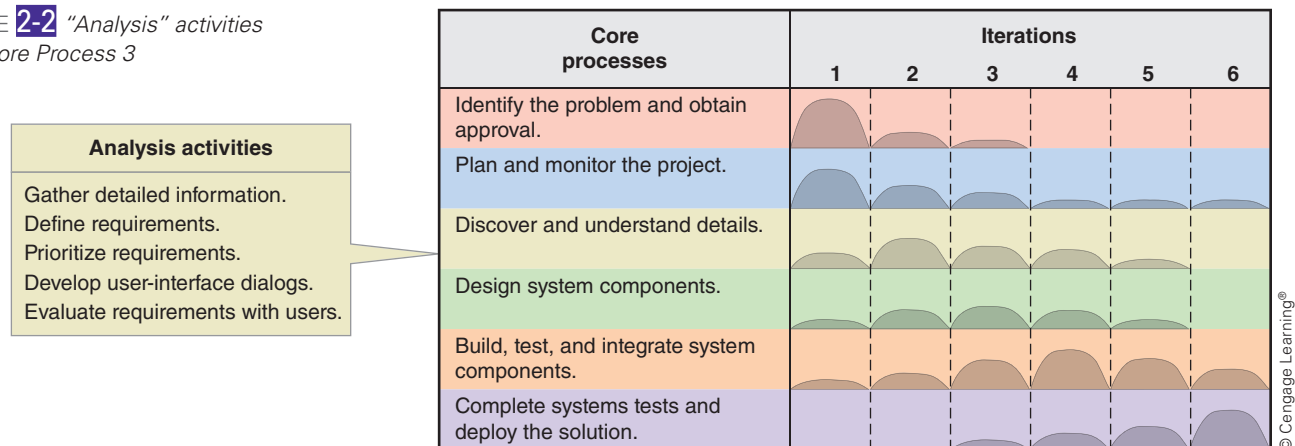
In later chapters, more details will be given about the capabilities of the new CSMS system. Assuming the project has been proposed, approved, and planned, the next section describes activities associated with the next step in the development process: systems analysis (SDLC Core Process 3). We will return to project management and project planning later in the text.

■ Systems Analysis Activities

The callout on the left side of **Figure 2-2** lists the activities of the third core process, which is to discover and understand the details. This core process also goes by the name *systems analysis*. The activities are as follows:

- Gather detailed information.
- Define requirements.
- Prioritize requirements.
- Develop user-interface dialogs.
- Evaluate requirements with users.

FIGURE 2-2 “Analysis” activities from Core Process 3



By completing these activities, the analyst defines in great detail what the information system needs to accomplish to provide the organization with the desired benefits.

Although we concentrate only on analysis activities in this chapter, keep in mind that they are usually intermixed with design, implementation, and other activities during the system development life cycle. For example, as shown in Figure 2-2, analysis activities are most intensive in the second iteration but occur in varying degrees during all project iterations except the last. This pattern is typical because an analyst often concentrates on one part of a system, defining requirements only for that part and simultaneously designing and implementing software to satisfy those requirements. Organizing development activities in this iterative manner enables development to be broken into smaller steps and helps users to validate requirements by testing and observing parts of the functional system. The following sections briefly describe analysis activities, and the remainder of this chapter expands the discussion of information gathering and defining system requirements.

■ Gather Detailed Information

Systems analysts obtain information from people who will be using the system, either by interviewing them or by watching them work. In short, analysts need to talk to nearly everyone who will use the new system or has used similar systems, and they must read nearly everything available about the existing system. Specifically, they obtain additional information by reviewing planning documents and policy statements; study existing systems, including their documentation; and obtain additional information by looking at what other companies (particularly vendors) have done when faced with a similar business need. Finally, they try to understand an existing system by identifying and understanding activities of all the current and future users by identifying all present and future locations where work occurs and all system interfaces with other systems, both inside and outside the organization. Later in this chapter, we discuss how to identify and extract information from all these people.

Beginning analysts often underestimate how much there is to learn about the work the user performs. The analyst must become an expert in the business area the system will support. For example, if you are implementing an order-entry system, you need to become an expert on the way orders are processed (including related accounting procedures). If you are implementing a loan-processing system, you need to become an expert on the rules used for approving credit. If you work for a bank, you need to think of yourself as a banker. The most successful analysts become experts in their organization's main business.

■ Define Requirements

The analyst uses information gathered from users and documents to define requirements for the new system. System requirements include the functions the system must perform (functional requirements) and such related issues as user-interface formats and requirements for reliability, performance, and security (nonfunctional requirements). We further discuss the distinction between functional and nonfunctional requirements a bit later in this chapter.

Defining requirements is not just a matter of writing down facts and figures. Instead, the analyst creates models to record requirements, reviews the models with users and others, and refines and expands the models to reflect new or updated information. In Chapter 1, you learned about requirements models, such as use case diagrams, activity diagrams, and domain model class diagrams. Building and refining requirements models occupies much of the analyst's time. This chapter and the next two chapters explain in considerable depth how to create requirements models.

■ Prioritize Requirements

Once the system requirements are well understood, it is important to establish which requirements are most crucial for the system. Sometimes, users request additional system functions that are desirable but not essential. However, users and analysts need to ask themselves which functions are truly important and which are fairly important but not absolutely required. Again, an analyst who understands the organization and the work done by the users will have more insight toward answering these questions.

Why prioritize the functions requested by the users? Resources are always limited, and the analyst must always be prepared to justify the scope of the system. Therefore, it is important to know what is absolutely required. Unless the analyst carefully evaluates priorities, system requirements tend to expand as users make more suggestions (a phenomenon called *scope creep*). Requirements priorities also help to determine the number, composition, and ordering of project iterations. High-priority requirements are often incorporated into early project iterations so analysts and users have ample opportunity to refine those parts of the system. Furthermore, a project with many high-priority requirements will typically have many iterations.

■ Develop User-Interface Dialogs

In some cases, when a new system is replacing an old system that does similar work, users are usually quite sure about their requirements and the desired form of the user interface. In other cases, the system development project breaks new ground by automating functions that were previously performed manually or by implementing functions that were not performed in the past. In either case, users tend to be uncertain of many aspects of system requirements. Such requirements models as use cases, activity diagrams, and interaction diagrams can be developed based on user input, but it is often difficult for users to interpret and validate such abstract models.

In comparison, user validation of a user interface is much simpler and more reliable because the user can see and feel the system. To most users, the user interface is all that matters. Thus, developing user-interface dialogs is a powerful method of eliciting and documenting requirements. Analysts can develop user interfaces via abstract models, such as interaction diagrams and written dialogs (covered in more detail in later chapters), or they can develop storyboards or user-interface prototypes on the actual input/output devices that users will use (e.g., a computer monitor, iPad, or smartphone). A prototype interface can serve as a requirement and a starting point for developing a portion of the system. A user-interface prototype developed in an early iteration can be expanded in later iterations to become a fully functioning part of the system.

■ Evaluate Requirements with Users

Ideally, evaluating requirements with users and documenting the requirements are fully integrated. But in practice, users generally have other responsibilities besides developing a new system. Thus, analysts usually use an iterative process in which they elicit user input to model requirements, return to the user for additional input or validation, and then work alone to incorporate the new input and refine the models. Prototypes of user interfaces and other parts of the system may also be developed when “paper” models are inadequate or when users and analysts need to prove that chosen technologies will do what they are supposed to do. Also, if the system will include new or innovative technology, the users may need help visualizing the possibilities available from the new technology as they define what they require. Prototypes can fill that need. The processes of eliciting requirements, building models and prototypes, and evaluating them with users may repeat many times until requirements models and prototypes are complete and accurate.

■ What Are Requirements?

As you can see from the previous section, requirements and models that represent them are a key focus of analysis phase activities. Most of the analyst's time is devoted to requirements: gathering information about them, formalizing them by using models and prototypes, refining and expanding them, prioritizing them, and generating and evaluating alternatives. But to fully understand those activities, you need to answer a fundamental question: What are requirements?

■ System Requirements and FURPS

System requirements are all the activities the new system must perform or support and the constraints that the new system must meet. Generally, analysts divide system requirements into two categories: functional and nonfunctional requirements. **Functional requirements** are the activities that the system must perform (i.e., the business uses to which the system will be applied). For example, if you are developing a payroll system, the required business uses might include such functions as “generate electronic fund transfers,” “calculate commission amounts,” “calculate payroll taxes,” “maintain employee-dependent information,” and “report tax deductions to the IRS.” The new system must handle all these functions. Identifying and describing all these business uses require a substantial amount of time and effort because the list of functions and their dependencies can be very complex.

In Chapter 1, the functional requirements were defined by the list of use cases for the Tradeshaw System, but functional requirements are also based on the procedures and rules that an organization uses to run its business. That is why use case details must be captured and modeled, too. Sometimes, they are well documented and easy to identify and describe. An example might be the following: “All new employees must fill out a W-4 form to enter information about their tax withholding in the payroll system.” Other business rules might be more obtuse or difficult to find. An example from RMO might be the following: “Air shipping charges are reduced by 50 percent for orders over \$200 that weigh less than two pounds.” Discovering such rules is critical to the final design of the system. If this rule were not discovered, customers who had relied on it in the past might become angry. Modifying the system after customers start complaining is much more difficult and expensive than building in the rule from the start.

Nonfunctional requirements are characteristics of the system other than those activities it must perform or support. It is not always easy to distinguish functional from nonfunctional requirements. One way to do so is to use a framework for identifying and classifying requirements. There have been many such frameworks developed over time; the most widely used today is referred to as FURPS (see [Figure 2-3](#)). **FURPS** is an acronym that stands for *functional, usability, reliability, performance, and security*. The *F* in FURPS is equivalent to the functional requirements defined previously. The remaining categories (URPS) describe nonfunctional requirements as follows:

- **Usability requirements** describe operational characteristics related to users, such as the user interface, related work procedures, online help, and documentation. For example, the user interface for a smartphone app should behave similarly to other apps when responding to such gestures as two-finger slides, pinching, and expanding. Additional requirements might include menu format, color schemes, use of the organization's logo, and multilanguage support.
- **Reliability requirements** describe the dependability of a system—how often a system exhibits such behaviors as service outages and incorrect processing and how it detects and recovers from those problems.

System requirements all the activities the new system must perform or support and the constraints that the new system must meet (functional + nonfunctional)

Functional requirements the activities the system must perform to support the users' work

nonfunctional requirements required system characteristics other than the activities it must perform or support

FURPS an acronym that stands for functional, usability, reliability, performance, and security requirements

usability requirements the requirements for operational characteristics related to users, such as the user interface, related work procedures, online help, and documentation

reliability requirements the requirements that describe system dependability

FIGURE 2-3 *FURPS acronym for functional, usability, reliability, performance, and security requirements*

Requirement categories	FURPS + categories	Example requirements
Functional	Functions	Business rules and processes
Nonfunctional	Usability Reliability Performance Security	User interface, ease of use Failure rate, recovery methods Response time, throughput Access controls, encryption

© Cengage Learning®

performance requirements the requirements that describe operational characteristics related to measures of workload, such as throughput and response time

security requirements the requirements that describe how access to the application will be controlled and how data will be protected during storage and transmission

FURPS+ an extension of FURPS that includes design constraints as well as implementation, system interface, physical, and supportability requirements

- **Performance requirements** describe operational characteristics related to measures of workload, such as throughput and response time. For example, the client portion of a system might be required to have a .5 second response time to all button presses, and the server might need to support 100 simultaneous client sessions (with the same response time).
- **Security requirements** describe how access to the application will be controlled and how data will be protected during storage and transmission. For example, the application might be password protected, encrypt locally stored data with 1024-bit keys, and use secure HTTP for communication among client and server nodes.

■ Additional Requirements Categories

FURPS+ is an extension of FURPS that adds additional categories, including design constraints as well as implementation, system interface, physical, and supportability requirements—all these additional categories summarized by the plus sign. Here are short descriptions of each category:

- *Design constraints* describe restrictions to which the hardware and software must adhere. For example, a cell phone application might be required to use the Android operating system, consume no more than 30 MB of flash memory storage, consume no more than 10 MB of system memory while running, and operate on CPUs rated at 1 GHz or higher.
- *Implementation requirements* describe constraints such as required programming languages and tools, documentation method and level of detail, and a specific communication protocol for distributed components.
- *Interface requirements* describe interactions among systems. For example, a financial reporting system for a publicly traded company in the United States must generate data for the Securities and Exchange Commission (SEC) in a specific XML format. The system might also supply data directly to stock exchanges and bond rating agencies, and automatically generate Twitter messages, RSS feeds, and Facebook updates.
- *Physical requirements* describe such characteristics of hardware as size, weight, power consumption, and operating conditions. For example, a system that supports battlefield communications might have such requirements as weighing less than 200 grams, being no larger than 5 centimeters cubed, and operating for 48 hours on a fully charged 1200 milliwatt lithium ion battery.
- *Supportability requirements* describe how a system is installed, configured, monitored, and updated. For example, requirements for a game installed on a home PC might include automatic configuration to maximize performance on existing hardware, error reporting, and download of updates from a support server.

As with any set of requirements categories, FURPS+ has some gray areas and some overlaps among its categories. For example, is a requirement that a battlefield communications device survive immersion in water and operate across a temperature range of -20°C to 50°C a performance or physical requirement?

Is a restriction to use no more than 100 megabytes of memory a performance or design requirement? Is a requirement to secure communication between workstations and servers with 1024-bit encryption a performance, design, or implementation requirement? What is important is that all requirements be identified and precisely stated early in the development process and that inconsistencies or trade-offs among them be resolved.

■ Stakeholders

Stakeholders persons who have an interest in the successful implementation of the system

internal stakeholders persons within the organization who interact with the system or have a significant interest in its operation or success

external stakeholders persons outside the organization's control and influence who interact with the system or have a significant interest in its operation or success

operational stakeholders persons who regularly interact with a system in the course of their jobs or lives

Stakeholders are your primary source of information for system requirements. **Stakeholders** are all the people who have an interest in the successful implementation of the system. Depending on the nature and scope of the system, this can be a small group, or a large, diverse group. For example, when implementing a comprehensive accounting system for a publicly traded corporation in the United States, the stakeholders include bookkeepers, accountants, managers and executives, customers, suppliers, auditors, investors, the SEC, and the Internal Revenue Service (IRS). Each stakeholder group interacts with the system in different ways, and each has a unique perspective on system requirements. Before gathering detailed information, the analyst identifies every type of stakeholder who has an interest in the new system and ensures that critical people from each stakeholder category are available to serve as the business experts.

One useful way to help identify all the interested stakeholders is to consider two dimensions by which they vary: internal stakeholders versus external stakeholders and operational stakeholders versus executive stakeholders (see **Figure 2-4**). **Internal stakeholders** are those within the organization who interact with the system or have a significant interest in its operation or success. You may be tempted to define internal stakeholders as employees of an organization, but some organizations—such as nonprofits and educational institutions—have internal users (e.g., volunteers and students) who are not employees. **External stakeholders** are those outside the organization's control and influence—although this distinction can also be fuzzy, such as when an organization's strategic partners (e.g., suppliers and shipping companies) interact directly with internal systems.

Operational stakeholders are those who regularly interact with a system in the course of their jobs or lives. Examples include accountants interacting with an accounting or billing system, factory supervisors interacting with a production scheduling system, customers interacting with an Internet storefront, and patients

FIGURE 2-4 Stakeholders of a comprehensive accounting system for a publicly traded company



executive stakeholders persons who don't interact directly with the system but who either use information produced by the system or have a significant financial or other interest in its operation and success

client a person or group that provides the funding for the system development project

who interact with a health-care Web site, Facebook page, or Twitter newsfeed. Operational users are a key source of requirements information, especially as it pertains to user interfaces and related functionality. **Executive stakeholders** are those who do not interact directly with the system, but who either use information produced by the system or have a significant financial or other interest in its operation and success. Examples include an organization's senior managers and board of directors, regulatory agencies, and taxing authorities.

Including such stakeholders in analysis activities is critical because the information they possess may not be obvious or widely known in the organization. In addition, system requirements imposed by executive stakeholders, especially external ones, often have significant legal and financial implications. For example, consider the potential effects of IRS regulations on an accounting system, or the effects of federal and state privacy laws on a social networking system.

Two other stakeholder groups that do not neatly fall into the categories just described deserve special attention. The **client** is the person or group that provides the funding for the project. In many cases, the client is senior management. However, clients may also be a separate group, such as a corporation's board of directors, executives in a parent company, or the board of regents of a university. The project team includes the client in its list of important stakeholders because the team must provide periodic status reviews to the client throughout development. The client or a direct representative on a steering or oversight committee also usually approves stages of the project and releases funds.

An organization's technical and support staff are also stakeholders in any system. The technical staff includes people who establish and maintain the computing environment of the organization. Support staff provide user training, troubleshooting, and related services. Both groups should provide guidance in such areas as programming language, computer platforms, network interfaces, and existing systems and their support issues. Any new system must fit within an organization's existing technology architecture, application architecture, and support environment. Thus, technical and support representatives are important stakeholders.

■ The Stakeholders for RMO

As a starting point for identifying CSMS stakeholders, it is helpful to develop a list of current CSS stakeholders, which include:

- Phone/mail sales order representatives
- Warehouse and shipping personnel
- Marketing personnel who maintain online catalog information
- Marketing, sales, accounting, and financial managers
- Senior executives
- Customers
- External shippers (e.g., UPS and FedEx)

Because the CSMS will take over existing functions of the CSS, the list of CSMS stakeholders includes all the stakeholders in the CSS list; however, there are some subtle differences. For example, the inclusion of social networking functions in the CSMS and the planned ability to share Mountain Bucks expands the definition of a customer. Whereas the old CSS was intended for use by potential customers visiting the Web site, the new system will interact with a much larger group of external stakeholders, including friends and family of existing customers and potentially all users of popular social networking sites. In essence, the stakeholder group "Customers" is much larger, more diverse, and includes people who have not purchased from RMO. Ensuring that the viewpoints and requirements of this diverse group are represented during analysis activities will be a considerable challenge.

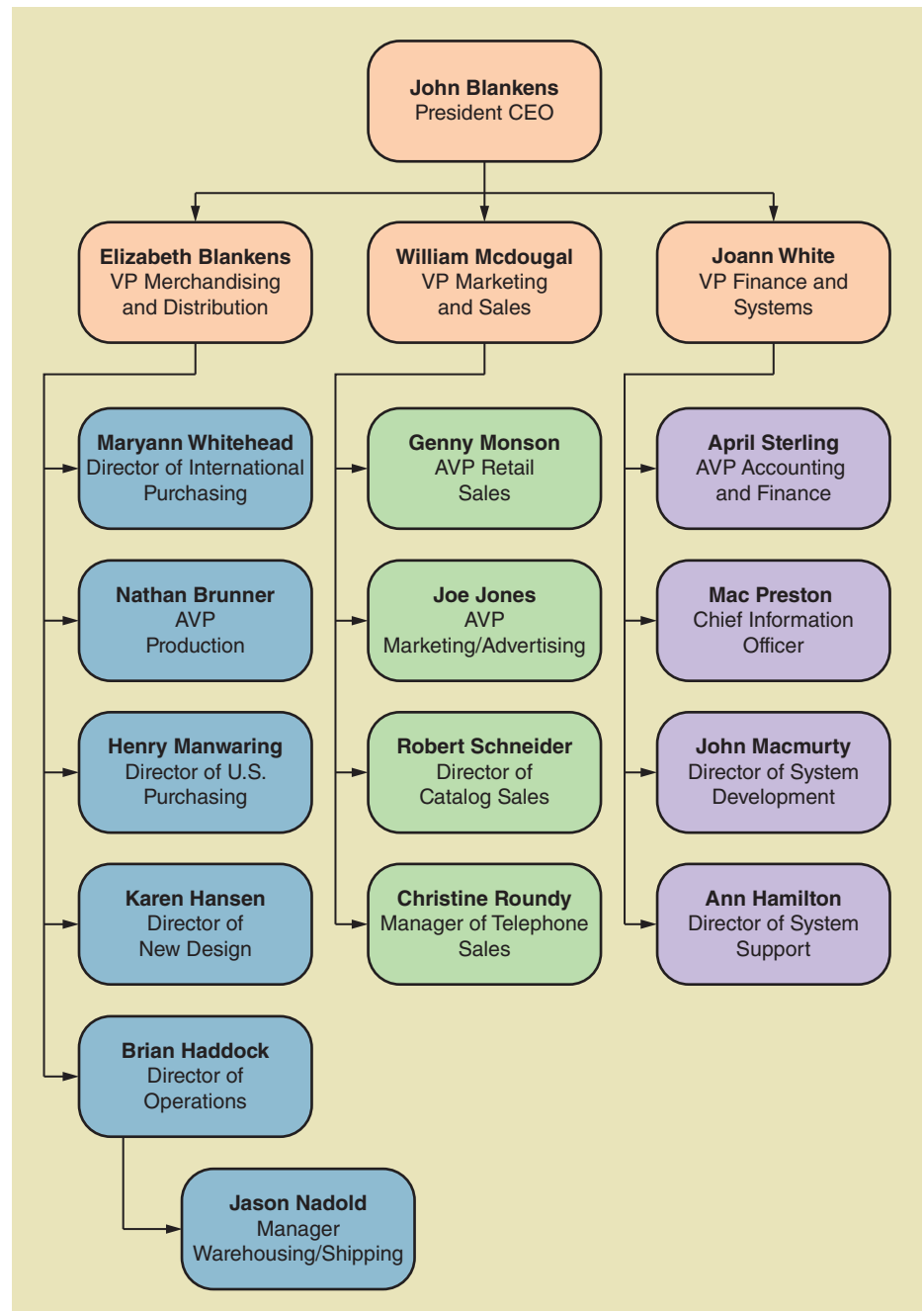
Expanded functionality for sales promotions with partner organizations creates an entirely new group of external stakeholders within those partner organizations. At this point, it is unclear whether that group will include operational

stakeholders, executive stakeholders, or both; nor is it known exactly how those stakeholders will interact with the system. Ensuring adequate input from this new group of stakeholders begins with defining precisely who they are.

RMO is a privately held company; John and Liz Blankens are the owners, and they hold two senior management positions. This is significant because the key operational systems of any publicly traded company “inherit” many external stakeholders due to the flow of information from those systems to the organization’s financial reports. RMO is audited by an external accounting firm, primarily to ensure access to bank loans and other private financing.

As owners and senior managers, John and Liz are the primary clients, but so are other senior executives who form a collaborative decision-making body. In addition, existing technical and support staff are key stakeholders. **Figure 2-5** summarizes the internal managerial stakeholders in the form of an organization chart.

FIGURE 2-5 RMO management stakeholders involved in the CSMS project



■ Information-Gathering Techniques

There are many ways that information about the system requirements can be collected. RMO often uses these standard information-gathering techniques:

- Interviewing users and other stakeholders
- Distributing and collecting questionnaires
- Reviewing inputs, outputs, and documentation
- Observing and documenting business procedures
- Researching vendor solutions
- Collecting active user comments and suggestions

All these methods have proven to be effective, although some are more efficient than others. In most cases, analysts combine methods to increase their effectiveness and efficiency and to provide a comprehensive fact-finding approach.

■ Interview Users and Other Stakeholders

Interviewing users and other stakeholders is an effective way to understand business functions and business rules. Unfortunately, it is also the most time-consuming and resource-expensive option. In this method, systems analysts do the following:

- Prepare detailed questions
- Meet with individuals or groups of users
- Obtain and discuss answers to the questions
- Document the answers
- Follow up as needed in future meetings or interviews

Obviously, this process may take some time, so it usually requires multiple sessions with each of the users or user groups.

■ Question Themes

Whether in informal meetings, formal interviews, or as part of a questionnaire or survey, analysts ask questions. But which questions should analysts ask? **Figure 2-6** shows three major themes that guide the analysts when they are asking questions to define system requirements; it also shows sample questions that arise from those themes.

What Are the Business Processes? The analyst must obtain a comprehensive list of all the business processes. In most cases, the users provide answers in terms of the current system, so the analyst must carefully discern which of those functions are fundamental and which may possibly be eliminated with an improved system. For example, salesclerks might indicate that the first task they perform when a customer places an order is to check the customer’s credit history. In the new system, salesclerks might never need to perform that function; the system might perform the check automatically. The function remains a system requirement, but the method of carrying out the function and its timing is changed.

FIGURE 2-6 Themes for information-gathering questions

Theme	Questions to users
What are the business operations and processes?	What do you do?
How should those operations be performed?	How do you do it? What steps do you follow? How could they be done differently?
What information is needed to perform those operations?	What information do you use? What inputs do you use? What outputs do you produce?

© Cengage Learning®

How Are the Business Processes Performed? Determining how business processes are performed begins with the current system but gradually moves to the new system. The goal is to define how the new system should support the function rather than how it supports it now. The analyst must be able to help the user visualize new and more efficient approaches to performing the business processes made possible by new technology or processes.

What Information Is Required? Some information inputs are formal; others are informal. When questioning the user, the analyst should specifically ask about exceptions or unusual situations in order to identify additional (nonroutine) information requirements. In this theme and the previous one, detail is the watchword. An analyst must understand the nitty-gritty detail to develop a correct solution.

■ Question Types

Questions can be roughly divided into two types:

open-ended questions questions that encourage discussion or explanation

closed-ended questions questions that elicit specific facts

- **open-ended questions**—such as “How do you do this function?”—encourage discussion and explanation.
- **closed-ended questions**—such as “How many forms a day do you process?”—are used to get specific facts.

Generally, open-ended questions help to start a discussion and enable a large number of requirements to be uncovered fairly quickly. Note that all the questions in the previous section are open ended. A discussion that starts with open-ended questions usually shifts gradually to closed-ended questions that elicit or confirm specific details of a business process.

■ Focus of Questions—Current System or New?

A significant question that faces all analysts is how much effort to expend studying and documenting the existing system (if one exists). Excess attention to an existing system can consume considerable time and can result in simply updating that system with newer technology. As a result, no matter how inefficient the current system is, system developers simply reimplement the procedures that are already in place. On the other hand, if a new system inherits many or all of the requirements of an existing system, then an analyst risks missing important requirements through insufficient study of the existing system.

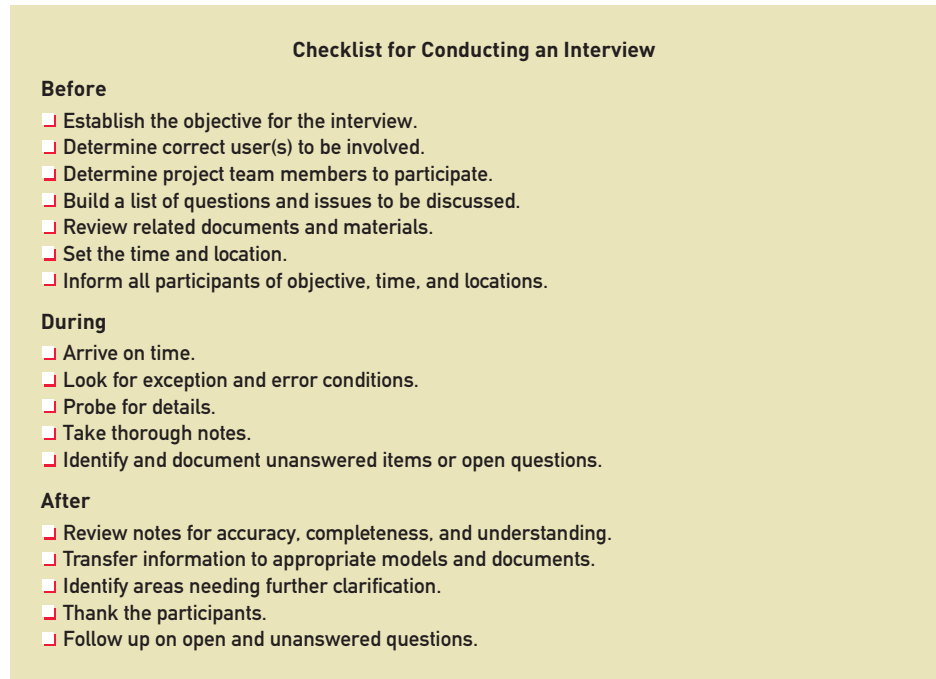
To minimize both risks, analysts must balance the review of current business functions with discovery of new system requirements. It is still critical to have a complete, correct set of system requirements, but in today’s fast-paced world, there is no time or money to review all the old systems and document all the inefficient procedures. In fact, in today’s development environment, one of the most valuable capabilities that a good system developer can bring is a new perspective to the problem.

■ Interview Preparation, Conduct, and Follow-Up

Figure 2-7 is a sample checklist that summarizes the major points to be covered; it is useful in preparing for, conducting, and following up an interview.

Preparing for the Interview Every successful interview requires preparation. The first and most important step in preparing for an interview is to establish its objective. In other words, you must determine what you want to accomplish with this interview. Write down the objective so it is firmly established in your mind. The second step is to determine which stakeholders should be interviewed. A small number of interviewees is generally best when the objective is narrow or of a fact-finding nature. Larger groups are better if the objective is more open ended, such as when generating and evaluating new ideas. However, it can be

FIGURE 2-7 Sample checklist to prepare for user interviews



difficult to manage a large group meeting well enough to ensure high-quality input from all participants. If possible, have at least two analysts involved in every interview, and have them compare notes afterward to ensure accuracy.

The next step is to prepare detailed questions to be used in the interview. Write down a list of specific questions, and prepare notes based on the forms or reports received earlier. Usually, you should prepare a list of questions that are consistent with the objective of the interview. Open-ended questions and closed-ended questions are both appropriate. Generally, open-ended questions help get the discussion started and encourage the user to explain all the details of the business process and the rules.

The last step is to make the final interview arrangements and then communicate those arrangements to all participants. A specific time and location should be established. If possible, a quiet location should be chosen to avoid interruptions. Each participant should know the objective of the meeting and, when appropriate, should have a chance to preview the questions or materials to be used. Interviews consume a substantial amount of time, and they can be made more efficient if each participant knows beforehand what is to be accomplished.

Conducting the Interview The usual rules of workplace meetings apply during stakeholder interviews: Plan ahead, arrive early, and ensure that the room is prepared and that needed resources are available. Limit the time of the interview for the benefit of the analyst(s) and stakeholder(s); stakeholders have other responsibilities, and the analysts can absorb only so much information at one time. It is better to have several shorter interviews than one long interview. A series of interviews provides an opportunity to absorb the material and then return for clarification later if needed.

Look for exception and error conditions. Look for opportunities to ask “what if” questions. “What if it doesn’t arrive? What if the signature is missing? What if the balance is incorrect? What if two orders are exactly the same?” The essence of good systems analysis is understanding all the “what ifs.” Make a conscious effort to identify all the exception conditions and then ask about them. More than any other skill, the ability to think of the exceptions will help you discover the detailed business rules. It is a hard skill to teach from a

textbook; experience will hone this skill. You will teach yourself this skill by conscientiously practicing it.

Probe for details. In addition to looking for exception conditions, the analyst must probe to ensure a complete understanding of all procedures and rules. One of the most difficult skills to learn as a new systems analyst is how to get enough details. Frequently, it is easy to get a general overview of how a process works; but do not be afraid to ask detailed questions until you thoroughly understand how the process works and what information is used. You cannot do effective systems analysis by glossing over the details.

Take careful notes. It is a good idea to take handwritten notes. Usually, tape recorders make users nervous. However, taking notes signals that you think the information you are obtaining is important, and the user is complimented. If two analysts conduct each interview, they can compare notes later. Identify and document in your notes any unanswered questions or outstanding issues that were not resolved. A good set of notes provides the basis for building the analysis models and for establishing a basis for the next interview session.

Figure 2-8 is a sample agenda for an interview session. Obviously, you do not need to conform exactly to a particular agenda. However, as with the interview checklist shown in Figure 2-7, this figure will help prod your memory on issues and items that should be discussed in an interview. Make a copy and use it. As you develop your own style, you can modify the checklist to the way you like to work.

FIGURE 2-8 Sample interview session agenda with follow-up information

Discussion and Interview Agenda
<p>Setting</p> <p>Objective of Interview <i>Determine processing rules for sales commission rates</i></p> <p>Date, Time, and Location <i>April 21, 2016, at 9:00 a.m. in William McDougal's office</i></p> <p>User Participants (names and titles/positions) <i>William McDougal, vice president of marketing and sales, and several of his staff</i></p> <p>Project Team Participants <i>Mary Ellen Green and Jim Williams</i></p>
<p>Interview/Discussion</p> <ol style="list-style-type: none"> 1. <i>Who is eligible for sales commissions?</i> 2. <i>What is the basis for commissions? What rates are paid?</i> 3. <i>How is commission for returns handled?</i> 4. <i>Are there special incentives? Contests? Programs based on time?</i> 5. <i>Is there a variable scale for commissions? Are there quotas?</i> 6. <i>What are the exceptions?</i>
<p>Follow-Up</p> <p>Important decisions or answers to questions <i>See attached write-up on commission policies</i></p> <p>Open items not resolved with assignments for solution <i>See Item numbers 2 and 3 on open items list</i></p> <p>Date and time of next meeting or follow-up session <i>April 28, 2016, at 9:00 a.m.</i></p>

Following up the Interview Follow-up is an important part of each interview. The first task is to absorb, understand, and document the information that was obtained. Generally, analysts document the details of the interview by constructing models of the business processes and writing textual descriptions of nonfunctional requirements. These tasks should be completed as soon after the interview as possible and the results distributed to the interview participants for validation. If the modeling methods are complex or unfamiliar to the users, the analyst should schedule follow-up meetings to explain and verify the models, as described in the last section of this chapter.

During the interview, you probably asked some “what if” questions that the users could not answer. They are usually policy questions raised by the new system that management has not considered before. It is extremely important that these questions not get lost or forgotten. **Figure 2-9** shows a sample table for tracking outstanding or unresolved issues for RMO. The table includes questions posed by users or analysts and responsibilities assigned for resolving the issues. If several teams are working, a combined list can be maintained. Other columns that might be added to the list are an explanation of the problem’s resolution and the date resolved.

Finally, make a list of new questions based on areas that need further elaboration or that are missing information. This list will prepare you for the next interview.

■ Distribute and Collect Questionnaires

Questionnaires enable analysts to collect information from a large number of stakeholders. Even if the stakeholders are widely distributed geographically, they can still help define requirements through questionnaires. Questionnaires are often used to obtain preliminary insight into stakeholder information needs, which helps to determine areas that need further research using other methods.

Figure 2-10 is a sample questionnaire showing three types of questions. The first part has closed-ended questions to determine quantitative information. The second part consists of opinion questions in which respondents are asked whether they agree or disagree with the statement. Both types of questions are useful for tabulating and determining quantitative averages. The third part requests an explanation of a procedure or problem. Questions such as these are good as a preliminary investigation to help direct further fact-finding activities.

Questionnaires are not well suited to helping you learn about processes, workflows, or techniques. Open-ended questions such as “How do you do this process?” are best answered by using interviews or observation. Although a questionnaire can contain a very limited number of open-ended questions, stakeholders frequently do not return questionnaires that contain many open-ended questions.

FIGURE 2-9 Sample open-items list

ID	Issue title	Date identified	Target end date	Responsible project person	User contact	Comments
1	Partial shipments	6-12-2016	7-15-2016	Jim Williams	Jason Nadold	Ship partials or wait for full shipment?
2	Returns and commissions	7-01-2016	9-01-2016	Jim Williams	William McDougal	Are commissions recouped on returns?
3	Extra commissions	7-01-2016	8-01-2016	Mary Ellen Green	William McDougal	How to handle commissions on special promotions?

© Cengage Learning®

FIGURE 2-10 Sample questionnaire

RMO Questionnaire

This questionnaire is being sent to all telephone-order sales personnel. As you know, RMO is developing a new customer support system for order taking and customer service.

The purpose of this questionnaire is to obtain preliminary information to assist in defining the requirements for the new system. Follow-up discussions will be held to permit everybody to elaborate on the system requirements.

Part I. Answer these questions based on a typical four-hour shift.

1. How many phone calls do you receive? _____
2. How many phone calls are necessary to place an order for a product? _____
3. How many phone calls are for information about RMO products, that is, questions only? _____
4. Estimate how many times during a shift customers request items that are out of stock. _____
5. Of those out-of-stock requests, what percentage of the time does the customer desire to put the item on back order? _____%
6. How many times does a customer try to order from an expired catalog? _____
7. How many times does a customer cancel an order in the middle of the conversation? _____
8. How many times does an order get denied due to bad credit? _____

Part II. Circle the appropriate number on the scale from 1 to 7 based on how strongly you agree or disagree with the statement.

Question	Strongly Agree				Strongly Disagree		
It would help me do my job better to have longer descriptions of products available while talking to a customer.	1	2	3	4	5	6	7
It would help me do my job better if I had the past purchase history of the customer available.	1	2	3	4	5	6	7
I could provide better service to the customer if I had information about accessories that were appropriate for the items ordered.	1	2	3	4	5	6	7
The computer response time is slow and causes difficulties in responding to customer requests.	1	2	3	4	5	6	7

Part III. Please enter your opinions and comments.

Please briefly identify the problems with the current system that you would like to see resolved in a new system.

■ Review Inputs, Outputs, and Procedures

There are two sources of information about inputs, outputs, and procedures. One source is external to the organization—industry-wide professional organizations and other companies. It may not be easy to obtain information from other companies, but they are a potential source of important information. Sometimes, industry journals and magazines report the findings of “best practices” studies. The project team would be negligent in its duties if its members were not familiar with best practice information.

The second source of inputs, outputs, and procedures includes existing business documents and procedure descriptions within the organization. Reviewing


internal documents and procedures serves two purposes. First, it is a good way to get a preliminary understanding of the processes. Second, existing inputs, outputs, and documents can serve as visual aids for the interview and as the working documents for discussion (see **Figure 2-11**). Discussion can focus on a specific input or output, its objective, its distribution, and its information content. The discussion should also include specific business events that initiate the use of an input or generation of an output. Several different business events might require the same form, and specific information about the event and the business process is critical. It is always helpful to have screens and forms that have been filled out with real information to ensure that the analyst obtains a correct understanding of the data content.

Reviewing the documentation of existing procedures helps identify business rules that may not come up in the interviews. Analyzing formal procedure documentation also helps reveal discrepancies and redundancies in the business processes. However, procedure documents frequently are not kept up to date, and they commonly include errors. To ensure that the assumptions and business rules that derive from the existing documentation are correct, analysts should review them with the users.

■ Observe and Document Business Processes

Firsthand experience is invaluable to understand exactly what occurs within business processes. More than any other activity, observing a business process in action will help you understand the business functions. However, while observing existing processes, you must also be able to visualize the new system's associated business processes. That is, as you observe the current business processes to understand the fundamental business needs, you should never forget that the processes could and often should change to be made more efficient. Do not get locked into believing there is only one way of performing the process.

FIGURE 2-11 RMO mail-order form used as a visual aid during an interview



Ridgeline Mountain Outfitters—Customer Order Form

Name and address of person placing order.
(Please verify your mailing address and make correction below.)

Order Date / /

Name _____

Address _____ Apt. No. _____

City _____ State _____ Zip _____

Phone: Day () _____ Evening () _____

Gift Order or Ship To: (Use only if different from address at left.)

Name _____

Address _____ Apt. No. _____

City _____ State _____ Zip _____

Gift Address for this Shipment Only Permanent Change of Address

Gift Card Message _____

Delivery Phone () _____

Item No.	Description	Style	Color	Size	Sleeve Length	Qty	Monogram	Style	Price Each	Total

Method of Payment

Check/Money Order Gift Certificate(s) AMOUNT ENCLOSED \$ _____

American Express MasterCard VISA Other

Account Number

MO YR /
Expiration Date

Signature _____

MERCHANDISE TOTAL _____

Regular FedEx shipping \$4.50 per U.S. delivery address
(Items are sent within 24 hours for delivery in 2 to 4 days)

Please add \$4.50 per each additional U.S. delivery address _____

FedEx Standard Overnight Service _____

Any additional freight charges _____

International Shipping (see shipping information on back) _____

You can observe a business process in many ways, ranging from a quick walk-through of an office or plant to doing the work yourself. A quick walk-through gives a general understanding of the layout of the office, the need for and use of computer equipment, and the general workflow. Spending several hours observing users at their jobs helps you understand the details of using the computer system and carrying out business functions. Being trained as a user and actually doing the job enables you to discover the difficulties of learning new procedures, the importance of a system that is easy to use, and the stumbling blocks and bottlenecks of existing procedures and information sources.

It is not necessary to observe all processes at the same level of detail. A quick walk-through may be sufficient for one process, whereas a process that is critical or more difficult to understand might require an extended observation period. If you remember that the objective is a complete understanding of the business processes and rules, you can assess where to spend your time to gain that understanding. As with interviewing, it is usually better if two analysts combine their efforts in observing procedures.

Observation often makes the users nervous, so you need to be as unobtrusive as possible. You can put users at ease in several ways, such as by working alongside a user or observing several users at once. Common sense and sensitivity to the needs and feelings of the users will usually result in a positive experience.

■ Research Vendor Solutions

Many of the problems and opportunities that companies want to address with new information systems have already been solved by other companies. In addition, consulting firms often have experience with the same problems, and software firms may have packaged solutions for a particular business need. Taking advantage of existing knowledge or solutions can avoid costly mistakes and save time and money.

There are three positive contributions and one danger in exploring existing solutions. First, researching existing solutions will frequently help users generate new ideas for how to better perform their business functions. Seeing how someone else solved a problem and applying that idea to the culture and structure of the existing organization will often provide viable alternative solutions for business needs.

Second, some of these solutions are excellent and state of the art. Without including this research, the development team may create a system that is obsolete even before it is designed. Companies need solutions that not only solve basic business problems, but that are up to date with competitive practices.

Third, it is often cheaper and less risky to buy a solution rather than to build it. If the solution meets the needs of the company and can be purchased, then that is usually a safer, quicker, and less-expensive route.

The danger in exploring existing solutions is that the users and even the systems analysts may want to buy one of the alternatives immediately. But if a solution, such as a packaged software system, is purchased too early in the process, the company's needs may not be thoroughly investigated. Too many companies have purchased systems only to find out later that they only support half the functions that were needed. Do not rush into a purchase decision until requirements are fully defined and all viable alternatives have been thoroughly investigated.

■ Collect Active User Comments and Suggestions

As discussed in Chapter 1 and earlier in this chapter, system development normally proceeds with analysis, design, and other activities spread across multiple iterations. Portions of the system are constructed and tested during each

iteration. Users and other stakeholders perform the initial testing of system functions during the iteration in which those functions are implemented. They also test and use those same functions during later iterations.

User feedback from initial and later testing is a valuable source of requirements information. Yet, interviews, discussions, and model reviews are an imperfect way of eliciting complete and accurate requirements. The phrase “I’ll know it when I see it” applies well to requirements definition. Users often cannot completely or accurately state their requirements until they can interact with a live system that implements those requirements. Based on those interactions, users can develop concrete suggestions for improvement and identify missing or poorly implemented requirements.

■ Models and Modeling

Modeling is an important part of systems analysis and design. You saw many different models in Chapter 1 for the Tradeshow System. Analysts build models to describe system requirements and use those models to communicate with users and designers. By developing a model and reviewing it with a user, an analyst demonstrates an understanding of the user’s requirements. If the user spots errors or omissions, they are incorporated into the model before it becomes the basis for subsequent design and implementation activities. **Figure 2-12** summarizes the key reasons for building and using models.

Designers construct high-level and detailed models to describe system components and their interactions. Design models serve as a scratch pad for evaluating design alternatives and as a way to communicate the final design to programmers, vendors, and others who will build, acquire, and assemble components to create the final system. In general, models built during one SDLC activity are “consumed” during other activities.

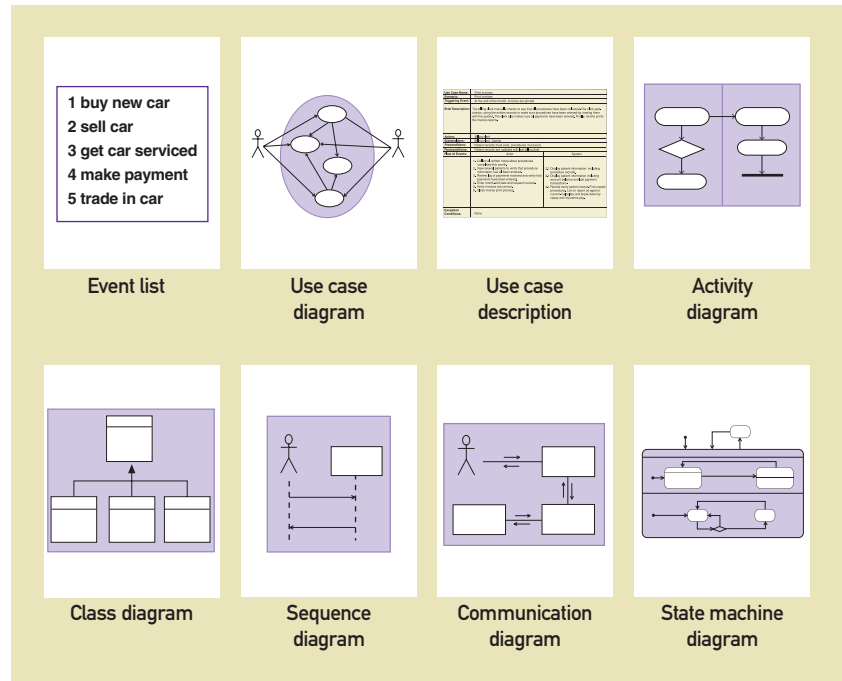
A **model** is a representation or abstraction of some aspect of the system being built. There are dozens of different models that an analyst or designer might develop and use (see **Figure 2-13**). Although this book emphasizes models and techniques for creating models, it is important to remember that system projects vary in the number of models required and in their formality. Smaller, simpler system projects will not need models showing every system detail, particularly when the project team has experience with the type of system being built. Sometimes, the key models are created informally in a few hours. Although models are often created by using specialized software tools, useful and important models are sometimes drawn quickly over lunch on a paper napkin or in an airport waiting room on the back of an envelope! As with any development activity, an iterative approach is used for creating models. The first draft of a model has some, but not all details worked out. The next iteration might fill in more details or correct previous misconceptions.

model representation or abstraction of some aspect of a system

FIGURE 2-12 *Reasons for modeling*

- ▣ Learning from the modeling process
- ▣ Reducing complexity by abstraction
- ▣ Remembering all the details
- ▣ Communicating with other development team members
- ▣ Communicating with a variety of users and stakeholders
- ▣ Documenting what was done for future maintenance/enhancement

FIGURE 2-13 Some analysis and design models



© Cengage Learning®

textual models text-based system models such as memos, reports, narratives, and lists

graphical models system models that use pictures and other graphical elements to create a diagram

mathematical models system models that describes requirements numerically or as mathematical expressions

Unified Modeling Language (UML) a standard set of information system model constructs and notations defined by the Object Management Group

Analysis and design models can be grouped into three generic types:

- **Textual models.** Analysts use such **textual models** as memos, reports, narratives, and lists to describe requirements that are detailed and are difficult to represent in other ways. The event list and use case descriptions (see icons shown in Figure 2-13) are two examples of textual models. Narrative description is often the best way to initially record information gathered verbally from stakeholders, such as during an interview.
- **Graphical models.** **Graphical models** make it easier to understand complex relationships that are difficult to follow when described as a list or narrative. Recall the old saying that a picture is worth a thousand words. In system development, a carefully constructed graphical model might be worth a million words! Some graphical models actually look similar to a real-world part of the system, such as a screen design or a report layout design. However, the graphical models developed during analysis activities typically represent more abstract things, such as external agents, processes, data, objects, messages, and connections.
- **Mathematical models.** **Mathematical models** are one or more formulas that describe technical aspects of a system. Analysts often use mathematical models to represent functional requirements for scientific and engineering applications and occasionally use them to describe business system requirements in areas such as accounting and inventory control. Analysts and designers use mathematical models to describe nonfunctional requirements and operational parameters such as network throughput or database query response time.

Many graphical models used in system development are drawn according to the notation specified by the **Unified Modeling Language (UML)**. In Figure 2-13, the use case diagram, class diagram, activity diagram, sequence diagram, communication diagram, and state machine diagram are UML graphical models. UML is the standard set of model constructs and notations defined by the Object Management Group (OMG), a standards organization for system development. By using UML, analysts and end users are able to depict and

understand a variety of specific diagrams used in a system development project. Prior to UML, there was no standard, so diagrams could be confusing, and they varied from company to company (and from book to book).

In later chapters, you will learn how to develop many of these analysis and design models. The first UML diagram covered in detail is the activity diagram.

■ Documenting Workflows with Activity Diagrams

workflow a sequence of work steps that completely handle one business transaction or customer request

activity diagram a UML diagram that describes user (or system) activities, the person or component that completes each activity, and the sequential flow of these activities

synchronization bar an activity diagram component that either splits a control path into multiple concurrent paths or recombines concurrent paths

swimlane an activity diagram component that divides the workflow activities into groups showing which agent performs which activity

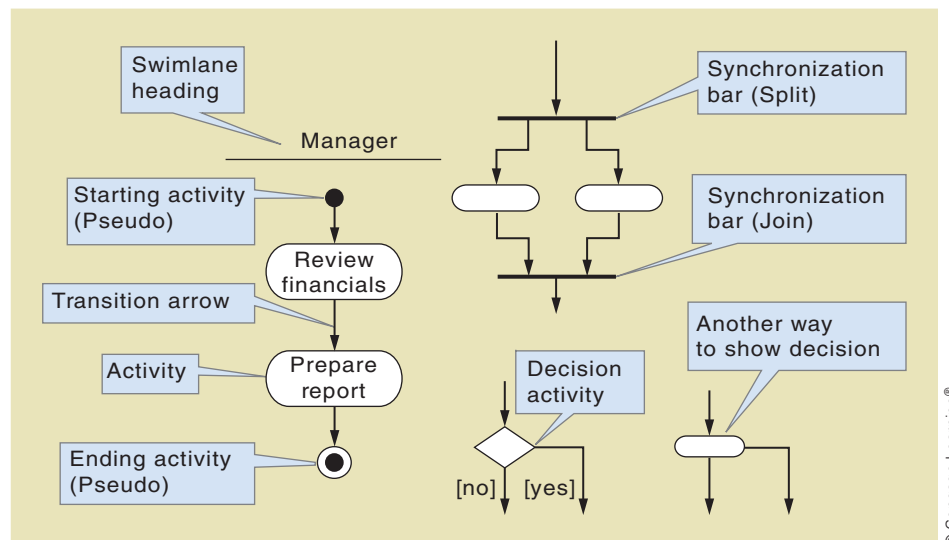
As you gather information about business processes, you will need to document your results as a workflow. A **workflow** is the sequence of work steps that complete one business transaction or customer request. Workflows may be simple or complex. Complex workflows can be composed of dozens or hundreds of work steps and may include participants from different parts of an organization.

One effective way to capture this information is with a UML activity diagram. An **activity diagram** describes the various user (or system) activities, the person or component that completes each activity, and the sequential flow of these activities.

Figure 2-14 shows the basic symbols used in an activity diagram. The flattened ovals represent the individual activities in a workflow. The connecting arrows represent the sequence of the activities. The black circles denote the beginning and the ending of the workflow. The diamond is a decision point at which the flow of the process will either follow one path or another. The heavy solid line is a **synchronization bar**, which either splits the path into multiple concurrent paths or recombines concurrent paths. The **swimlane** represents an agent who performs the activities. It is called a swimlane because each agent follows a path parallel with other agents in the workflow, as with swimmers in a swimming pool.

Figure 2-15 is an activity diagram that describes the order fulfillment process for the current RMO CSMS. Processing begins when the customer has completed the order checkout process and the warehouse begins order fulfillment. The diagram describes the back-and-forth flow of information and control between the order subsystem, inventory subsystem, warehouse(s), and shipper. The diagram is simplified because it omits many error-handling pathways, including what happens if enough item stock is on hand to fulfill only part of an order.

FIGURE 2-14 Activity diagram symbols



© Cengage Learning®

FIGURE 2-15 Simple activity diagram for order fulfillment after online checkout

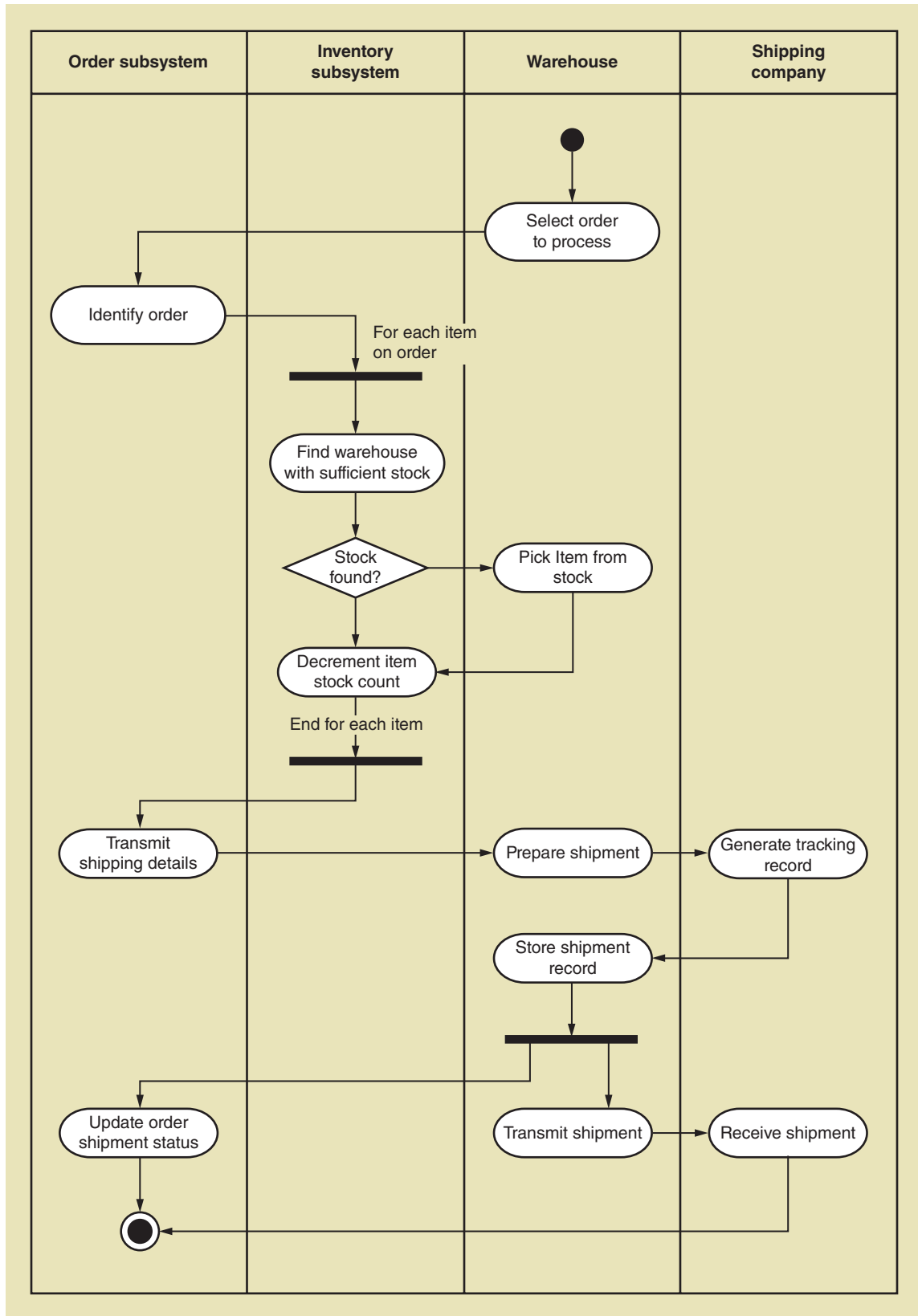
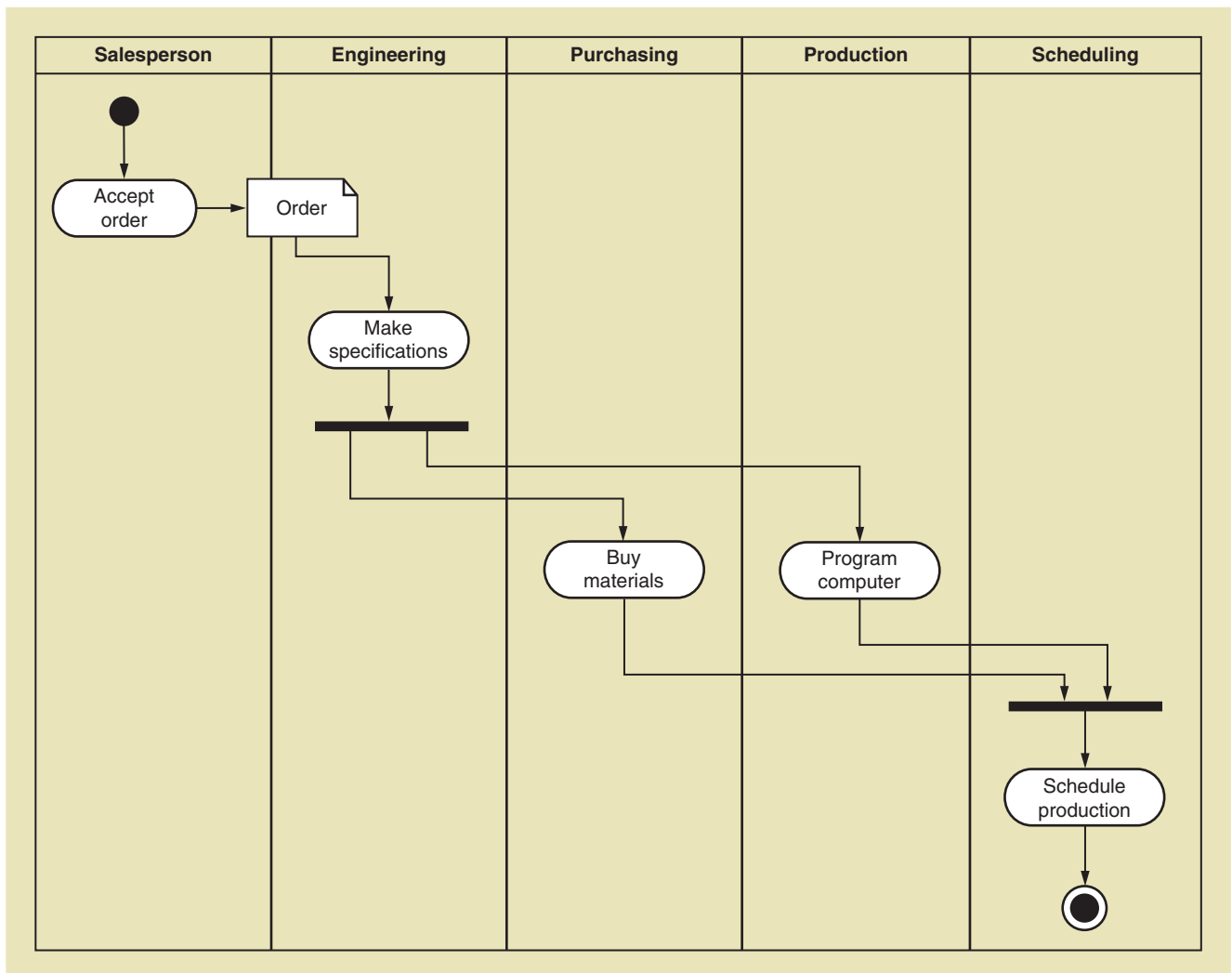


Figure 2-16 illustrates another workflow diagram, which demonstrates some new concepts. In this example, a customer is ordering a product that has to be manufactured to match customer specifications. To show that the salesperson sends the order to Engineering, the diagram uses a new symbol to emphasize the transmission of the document between Sales and Engineering. After Engineering develops the specifications, two concurrent activities happen: Purchasing orders the materials, and Production writes the program for the automated milling machines. These two activities are completely independent and can occur at the same time. Notice that one synchronization bar splits the path into two concurrent paths and that another synchronization bar reconnects them. Finally, Scheduling puts the order on the Production schedule.

Creating activity diagrams to document workflows is straightforward. The first step is to identify the agents to create the appropriate swimlanes. Next, follow the various steps of the workflow and then make appropriate ovals for the activities. Connect the activity ovals with arrows to show the workflow. Here are a couple guidelines:

- Use a decision symbol to represent an either/or situation—one path or the other path but not both. As a shorthand notation, you can merge an activity (by using an oval) and a decision (by using a diamond) into a single oval

FIGURE 2-16 Activity diagram showing concurrent paths



with two exit arrows, as indicated on the right in Figure 2-14. This notation represents a decision (either/or) activity. Wherever you have an activity that reads “verify” or “check,” you will probably require a decision—one for the “accept” path and one for the “reject” path.

- Use synchronization bars for parallel paths—situations in which both paths are taken. Include a beginning and an ending synchronization bar. You can also use synchronization bars to represent a loop, such as a “do while” programming loop. Put the bar at the beginning of the loop and then describe it as “for every.” Put another synchronization bar at the end of the loop with the description “end for every.”

CHAPTER SUMMARY

This chapter focuses on system requirements, which is the focus of the Core Process 3 (systems analysis) of the SDLC. There are five primary activities of systems analysis:

- Gather detailed information.
- Define requirements.
- Prioritize requirements.
- Develop user-interface dialogs.
- Evaluate requirements with users.

Functional requirements are those that explain the basic business functions that the new system must support. Nonfunctional requirements involve the system’s objectives with regard to usability, reliability, performance, and security.

Stakeholders include internal and external users of the system and other persons or organizations that have a vested interest in the system.

Analysts use many techniques to gather information about requirements, including:

- Interviews
- Questionnaires
- Documentation, input and output reviews
- Process observation and documentation
- Vendor solution research
- Active comments and suggestions from users

Textual, graphical, and mathematical models are developed to document requirements and as an aid in evaluating requirements with users and other stakeholders. Models are created when gathering information about the system. They are also created when designing the system. Unified Modeling Language (UML) defines a collection of diagrams and constructs used for system modeling.

UML activity diagrams are used to model workflows, a technique often used as an early requirements model. Activity diagrams graphically model the steps of a business process and the participants who perform them. Other models and UML diagrams are covered in later chapters.

KEY TERMS

activity diagram

application architecture

client

closed-ended questions

executive stakeholders

external stakeholders

functional requirements

FURPS

FURPS+

graphical models

internal stakeholders

mathematical models

model

nonfunctional requirements

open-ended questions

operational stakeholders

performance requirements

reliability requirements

security requirements

stakeholders

swimlane

synchronization bar

system requirements

technology architecture

textual models

Unified Modeling Language (UML)

usability requirements

workflow

REVIEW QUESTIONS

1. List and briefly describe the five activities of systems analysis.
2. What are three types of models?
3. What is the difference between functional requirements and nonfunctional requirements?
4. Describe the steps in preparing for, conducting, and following up on an interview session.
5. What are the benefits of doing vendor research during information-gathering activities?
6. What types of stakeholders should you include in fact finding?
7. Describe the open-items list and then explain why it is important.
8. List and briefly describe the six information-gathering techniques.
9. What is the purpose of an activity diagram?
10. Draw and explain the symbols used on an activity diagram.
11. Explain why the Unified Modeling Language (UML) is important to use as a standard for creating information systems models.

PROBLEMS AND EXERCISES

1. Provide an example of each of the three types of models that might apply to designing a car, a house, and an office building.
2. One of the toughest problems in investigating system requirements is to make sure they are complete and comprehensive. How would you ensure that you get all the right information during an interview session?
3. One of the problems you will encounter during your investigation is *scope creep* (i.e., user requests for additional features and functions). Scope creep happens because users can have problems with the current system and the system investigation may be the first time anybody has listened to their needs. How do you keep the system from including new functions that expand the scope beyond the constraints imposed by the project schedule or budget?
4. What would you do if you got conflicting answers for the same procedure from two different people you interviewed? What would you do if one was a clerical person and the other was the department manager?
5. You have been assigned to resolve several issues on the open-items list, and you are having a hard time getting policy decisions from the user contact. How can you encourage the user to finalize these policies?
6. In the running case of RMO, assume that you have set up an interview with the manager of the Shipping Department. Your objective is to determine how shipping works and what the information requirements for the new system will be. Make a list of questions—open ended and closed ended—that you would use. Include any questions or techniques you would use to ensure you find out about the exceptions.
7. Develop an activity diagram based on the following narrative. Note any ambiguities or questions that you have as you develop the model. If you need to make assumptions, also note them.

The Purchasing Department handles purchase requests from other departments in the company. People in the company who initiate the original purchase request are the “customers” of the Purchasing Department. A case worker within the Purchasing Department receives the request and monitors it until it is ordered and received.

Case workers process requests for the purchase of products under \$1,500, write a purchase order, and then send it to the approved vendor. Purchase requests over \$1,500 must first be sent out for bid from the vendor that supplies the product. When the bids return, the case worker selects one bid and then writes a purchase order and sends it to the vendor.
8. Develop an activity diagram based on the following narrative. Note any ambiguities or questions that you have as you develop the model. If you need to make assumptions, also note them.

The Shipping Department receives all shipments on outstanding purchase orders. When the clerk in the Shipping Department receives a shipment, he or she finds the outstanding purchase order for those items. The clerk then sends multiple copies of the shipment packing slip. One copy goes to Purchasing, and the department updates its records to indicate that the purchase order has been fulfilled. Another copy goes to Accounting so a payment can be made. A third copy goes to the requesting in-house customer so he or she can receive the shipment.

After payment is made, the Accounting Department sends a notification to Purchasing. After the customer receives and accepts the goods, he or she sends notification to Purchasing. When Purchasing receives these other verifications, it closes the purchase order as fulfilled and paid.

9. Conduct a fact-finding interview with someone involved in a procedure that is used in a business or organization. This person could be someone at the university, in a small business in your neighborhood, in the student volunteer office at the university, in a doctor's or dentist's office, or in a volunteer organization. Identify a process, such as keeping student records, customer records, or

member records. Make a list of questions and then conduct the interview. Remember, your objective is to understand that procedure thoroughly (i.e., to become an expert on that single procedure).

10. Using RMO and the CSMS as your guide, develop a list of all the procedures that may need to be researched. You may want to think about the exercise in the context of your experience with such retailers as L.L. Bean, Lands' End, or Amazon.com. Check out the Internet marketing done on the retailers' Web sites and then think about the underlying business procedures that are required to support those sales activities. List the procedures and then describe your understanding of each.

CASE STUDY

John and Jacob, Inc.: Online Trading System

John and Jacob, Inc., is a regional brokerage firm that has been successful over the last several years. Competition for customers is intense in this industry. The large national firms have very deep pockets, with many services to offer clients. Severe competition also comes from discount and Internet trading companies. However, John and Jacob has been able to cultivate a substantial customer base from upper-middle-income clients in the northeastern United States. To maintain a competitive edge with its customers, John and Jacob is in the process of modernizing its online trading system. The modernization will add new features to the existing system and expand the range of interfaces beyond desktop and laptop computers to include tablet computers and smartphones. The system will add Twitter messaging in addition to continued support for traditional e-mail.

Edward Finnigan, the project manager, is in the process of identifying all the groups of people who should be included in the development of the system requirements. He is not quite sure exactly who should be included. Here are the issues he is considering:

- **Users.** The trading system will be used by customers and by staff in each of the company's 30 trading offices. Obviously, the brokers who are going to use the system need to have input, but how should this be done? Edward also is not sure what approach would be best to ensure that the requirements are complete without requiring tremendous amounts of time from the stakeholders. Including all the offices would increase enthusiasm and support for the system, but it would increase the time required to compile the information. Furthermore, involving more brokers would bring divergent opinions that would have to be reconciled.

- **Customers.** The trading system will also include trade order entry, investment analysis reports, trade confirmations, standard and customized reporting, and customer statements. Edward wonders how to involve John and Jacob customers in the development of system requirements. Edward is sensitive to this issue because many brokers have told him that their customers are unhappy with the current system, and customer complaints are sometimes posted to the public comments area of the current Web site. He would like to involve customers, but he does not know how.
- **Other stakeholders.** Edward knows he should involve other stakeholders to help define system requirements. He is not quite sure whom he should contact. Should he go to senior executives? Should he contact middle management? Should he include such back-office functions as accounting and investing? He is not quite sure how to get organized or how to decide who should be involved.

1. What is the best method for Edward to involve the brokers (users) in the development of the updated online trading system? Should he use a questionnaire? Should he interview the brokers in each of the company's 30 offices or would one or two brokers representing the entire group be better? How can Edward ensure that the information about requirements is complete, yet not lose too much time doing so?
2. Concerning customer input for the new system, how can Edward involve customers in the process? How can he interest them in participating? What methods can Edward use to ensure that the customers he involves are representative of John and Jacob's entire customer group?
3. As Edward considers what other stakeholders he should include, what are some criteria he should use? Develop some guidelines to help him build a list of people to include.

RUNNING CASE STUDIES

Community Board of Realtors®

The real estate business relies on an extensive amount of information used in the buying and selling of real property. Most communities of real estate agents and brokers have formed cooperative organizations to help consolidate and distribute information on the real estate profession, real estate trends, properties in the community, historical records of property sales, and current listings of properties for sale. These organizations are usually referred to as the Community Board of Realtors.

Research your local Community Board of Realtors to answer these questions:

1. Who are the stakeholders for the issues related to real estate in your community, and what are their main interests?
2. What types of information does the board collect and make available to its members and to the community?
3. Research the real estate industry in at least two countries other than the United States. For each of these countries, what are some of the cultural and legal issues that differ from those in the United States? If you were working on support for an international real estate cooperative system, in what ways would the information collection activity process be complicated?

The Spring Breaks 'R' Us Travel Service

Spring Breaks 'R' Us (SBRU) is an online travel service that books spring break trips to resorts for college students. Students have booked spring break trips for decades, but changes in technology have transformed the travel business in recent years. SBRU moved away from having campus reps with posted fliers and moved to the Web early on. The basic idea is to get a group of students to book a room at a resort for one of the traditional spring break weeks. SBRU contracts with dozens of resorts in key spring break destinations like Florida, Texas, the Caribbean, and Mexico. Its Web site shows information on each resort and includes prices, available rooms, and special features. Students can research and book a room, enter contract information, and pay deposits and final payments through the system. SBRU provides updated booking information, resort information updates, and travel information for booked students when they log in to the site.

The resorts also need access to information from SBRU. They need to know about their bookings for each week, the room types that are booked, and so forth. Before the spring break booking season starts, they need to enter information on their resorts, including prices and special features. Resorts need to be paid by SBRU for the bookings, and they need to be able to report and collect for damages caused by spring breakers during their stay.

SBRU has recently decided to upgrade its system to provide social networking features for students. It is currently researching possibilities and collecting information from prospective customers about desirable

features and functions. From the business standpoint, the idea is to increase bookings by enhancing the experience before, during, and after the trip.

1. Who are the stakeholders for SBRU? For each type of stakeholder, what aspects of the SBRU booking system are of particular interest?
2. What are the main functional requirements for the major subsystems (i.e., resort relations, student booking, accounting and finance, and social networking)?
3. Describe some usability requirements for students, booking interactions, and social networking interactions.
4. Assuming that social networking at the resorts will require wireless communication and connection to the Internet, what are some reliability requirements that resorts might be asked to maintain? What are some performance requirements? Is this a bigger issue because resorts are in international locations?
5. What are some security requirements? Is there any reason why students in Europe, Asia, or other locations could not book rooms through SBRU? What issues might be anticipated?
6. To collect information on functional requirements for the social networking subsystem, what are some techniques that might be used? Be specific and include some sample questions you might ask by using various techniques.

On the Spot Courier Services

As an employee of a large international courier and shipping service, Bill Wiley met with many companies that shipped and received packages almost every day. He was frequently asked if his company could deliver local packages on the same day. Over several months, he observed that there appeared to be a substantial need for courier services in the city in which he lived. He decided that he would form his own courier delivery company called On the Spot to fill this need.

Bill began by listing his mobile telephone number in the Yellow Pages. He also sent letters to all those companies that had requested same-day courier service that his prior company had not been able to serve. He hoped that through good service and word-of-mouth advertising that his business would grow. He also began other advertising and marketing activities to promote his services.

At first, Bill received delivery requests on his business mobile phone. However, it was not long before his customers were asking if he had a Web site where they could place orders for shipments. He knew that if he could get a Web presence that he could increase his exposure and help his business grow.

After he had been in business only a few short months, Bill discovered he needed to have additional help. He hired another person to help with the delivery and pickup of packages. It was good to see the business

grow, but another person added to the complexity of coordinating pickups and deliveries. With the addition of a new person, he could no longer “warehouse” the packages out of his delivery van. He now needed a central warehouse where he could organize and distribute packages for delivery. He thought that if his business grew enough to add one more delivery person that he would also need someone at the warehouse to coordinate the arrival and distribution of all the packages.

1. Who are the stakeholders for On the Spot? How involved should On the Spot’s customers be in system definition? As the business grows, who else might be potential stakeholders and interested in system functions?
2. If you were commissioned to build a system for Bill, how would you determine the requirements? Be specific in your answer. Make a list of the questions you need answered.
3. What technology and communication requirements do you see? What are the hardware requirements, and what kind of equipment will provide viable options to the system? What would you recommend to Bill?
4. What are the primary functional requirements for the system as described so far in the case?

Sandia Medical Devices

Medical monitoring technology has advanced significantly in the last decade. Monitoring that once required a visit to a health-care facility can now be performed by devices located in a patient’s home, or carried or worn at all times. Examples include measures of glucose level (blood sugar), pulse, blood pressure, and electrocardiogram (EKG). Measurements can be transmitted via telephone, Internet connection, and wireless data transmission standards, such as Bluetooth. A particularly powerful technology combination is a wearable device that records data periodically or continuously and transmits it via Bluetooth to a cell phone app. The cell phone app can inform the patient of problems and can automatically transmit data and alerts to a central monitoring application (see **Figure 2-17**).

Health-care providers and patients incur significant costs when glucose levels are not maintained within acceptable tolerances. Short-term episodes of very high or very low glucose often result in expensive visits to urgent care clinics or hospitals. In addition, patients with frequent but less severe episodes of high

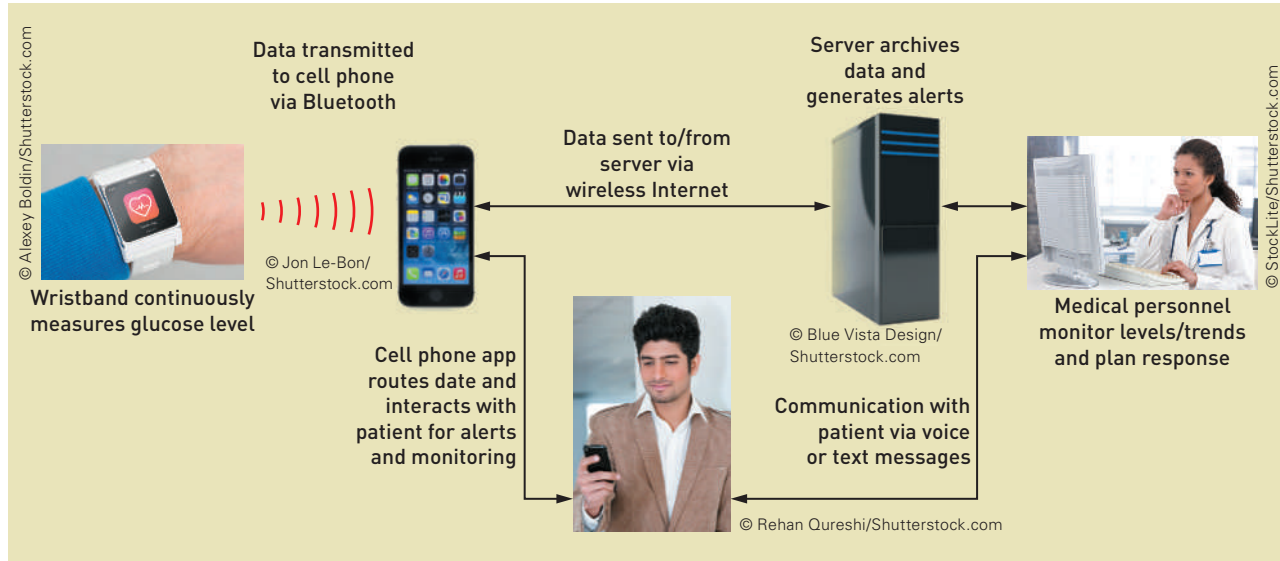
or low glucose are more susceptible to such expensive, long-term complications as vision, circulatory, and kidney problems.

Sandia Medical Devices (SMD), an Albuquerque manufacturer of portable and wearable medical monitoring devices, has developed a glucose monitor embedded in a wristband. The device is powered by body heat and senses glucose levels from minute quantities of perspiration. SMD is developing the Real-Time Glucose Monitoring (RTGM) device in partnership with New Mexico Health Systems (NMHS), a comprehensive health delivery service with patients throughout New Mexico. The system’s vision statement reads as follows:

RTGM will enable patients and their health-care providers to continuously monitor glucose levels, immediately identify short- and long-term medical dangers, and rapidly respond to those dangers in medically appropriate ways.

SMD will develop the initial prototype software for smartphones with Bluetooth capability running

FIGURE 2-17 Data movement among devices and users



the Google Android operating system. If successful, NMHS and its patients will have free use of the software and SMD will resell the software to other health systems worldwide.

1. Who are RTGM's stakeholders? Should NMHS's patients be included in defining the system requirements? Why or why not? Should RTGM interact with medical professionals other than physicians? Why or why not?
2. If you were the lead analyst for RTGM, how would you determine the requirements?

Be specific in your answer. List several questions you need answered.

3. What are the primary functional requirements for the system as described so far in the case?
4. Are the parameters for alerting patients and medical personnel the same for every patient? Can they vary over time for the same patient? What are the implications for the system's functional requirements?
5. Briefly describe some possible nonfunctional requirements for RTGM.

FURTHER RESOURCES

Soren Lauesen, *Software Requirements: Styles and Techniques*. Addison-Wesley, 2002.

Stan Magee, *Guide to Software Engineering Standards and Specifications*. Artech House, 1997.

Suzanne Robertson and James Robertson, *Mastering the Requirements Process*, Second Edition. Addison-Wesley, 2006.

Karl Wiegers, *Software Requirements*. Microsoft Press, 2003.

Karl Wiegers, *More About Software Requirements: Thorny Issues and Practical Advice*. Microsoft Press, 2006.

Ralph Young, *The Requirements Engineering Handbook*. Artech House, 2003.



Identifying User Stories and Use Cases

CHAPTER THREE

CHAPTER OUTLINE

- User Stories and Use Cases
- Use Cases and the User Goal Technique
- Use Cases and Event Decomposition
- Use Cases in the Ridgeline Mountain Outfitters Case

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- Explain why identifying user stories and use cases is the key to defining functional requirements
- Write user stories with acceptance criteria
- Describe the two techniques for identifying use cases
- Apply the user goal technique to identify use cases
- Apply the event decomposition technique to identify use cases
- Describe the notation and purpose for the use case diagram
- Draw use case diagrams by actor and by subsystem

OPENING CASE WAITERS ON CALL MEAL-DELIVERY SYSTEM

Waiters on Call is a restaurant meal-delivery service started in 2010 by Sue and Tom Bickford. The Bickfords worked for restaurants while in college and always dreamed of opening their own restaurant; unfortunately, the initial investment was always out of reach. The Bickfords noticed that many restaurants offer takeout food, and that some restaurants—primarily pizzerias—offer home-delivery service. However, many people they met seemed to want home delivery with a wider food selection.

Sue and Tom conceived Waiters on Call as the best of both worlds: a restaurant service without the high initial investment. They contracted with a variety of well-known restaurants in town to accept orders from customers and to deliver the complete meals. After preparing the meal to order, the restaurant charges Waiters on Call a wholesale price, and the customer pays retail plus a service charge and tip when the meals are delivered. Waiters on Call started modestly, with only two restaurants and one delivery driver working the dinner shift. Business rapidly expanded, until the Bickfords realized they needed a custom computer system to support their operations. They hired a consultant, Sam Wells, to help them define what sort of system they needed.

“What events happen when you are running your business that make you want to reach for a computer?” asked Sam. “Tell me about what usually goes on.”

“Well,” answered Sue, “when a customer calls in wanting to order, I need to record it and get the information to the right restaurant. I need to know which drivers are available to pick up the order, so I need drivers to call in and tell me when they are free. Perhaps this could be included as a smartphone or iPad app. Sometimes, customers call back wanting to change their orders, so I need to find the original order and notify the restaurant to make the change.”

“Okay, that’s great. Now, how do you handle the money?” queried Sam.

Tom jumped in. “The drivers get a copy of the bill showing the retail price directly from the restaurant when they pick up the meal. The bill should agree with our calculations. The drivers collect that amount plus a service charge. When drivers report in at closing, we

add up the money they have and compare it with the records we have. After all drivers report in, we need to create a deposit slip for the bank for the day’s total receipts. At the end of each week, we calculate what we owe each restaurant at the agreed-to wholesale price and send them a statement and check.”

“What other information do you need to get from the system?” continued Sam.

“It would be great to have some information at the end of each week about orders by restaurant and orders by area of town—things like that,” Sue said. “That would help us decide about advertising and restaurant contracts. Then, we need monthly statements for our accountant.”

Sam made some notes and sketched some diagrams as Sue and Tom talked. Then, after spending some time thinking about it, he summarized the situation for Waiters on Call. “It sounds to me like you need a system to use whenever these events occur”:

- A customer calls in to place an order, so you need to *Record an order*.
- A driver is finished with a delivery, so you need to *Record delivery completion*.
- A customer calls back to change an order, so you need to *Update an order*.
- A driver reports for work, so you need to *Sign in the driver*.
- A driver submits the day’s receipts, so you need to *Reconcile driver receipts*.

Sam continues, “Then, you need the system to produce information at specific points in time—for example, when it is time to,”

- *Produce an end-of-day deposit slip.*
- *Produce end-of-week restaurant payments.*
- *Produce weekly sales reports.*
- *Produce monthly financial reports.*

“Am I on the right track?”

Sue and Tom quickly agreed that Sam was talking about the system in a way they could understand. They were confident that they had found the right consultant for the job.

■ Overview

Chapter 2 described the systems analysis activities used in system development and introduced the tasks and techniques involved when completing the first analysis activity—gathering information about the system, its stakeholders, and its requirements. An extensive amount of information is required to properly define the system’s functional and nonfunctional requirements. This chapter, with Chapter 4 and Chapter 5, presents techniques for documenting

the functional requirements by creating a variety of models. These models are created as part of the analysis activity *Define requirements*, although remember that the analysis activities are actually done in parallel with design and implementation and in each iteration of the project.

In the Waiters on Call case, Sam Wells is working with the Bickfords to identify the functional requirements for the new system using the event decomposition technique. The sketch he was drawing was a use case diagram. You will learn about this technique and others that help identify user stories and use cases in this chapter.

■ User Stories and Use Cases

As you saw in Chapter 1, identifying user stories and use cases is a key task when defining functional requirements because these form the basis for the list of functions the system needs to carry out. Virtually all recent approaches to system development begin the requirements modeling activity with the concept of a user story or a use case. These two concepts are similar in that they focus on the goals of the user, and they show the list of functions at the appropriate level of detail. But they differ in the approach taken to identify them and in the amount of detail that is captured by the analyst. User stories are favored by highly Agile system development methodologies, and they are turned over to the programmer analyst much earlier than use cases are. The programmer analyst designs and codes each user story to discover needed details. The Agile development philosophy is to work directly with users and avoid doing too much documentation. In contrast, a use case approach traditionally meant analysts complete much documentation for each use case, focusing on detailed steps carried out by the user and the system. In practice, use cases can also be very brief for Agile development.

user story one short sentence in the everyday language of the end user that states what a user does as part of his or her work

A **user story** is usually one short sentence in the everyday language of the end user that states what a user does as part of his or her work. In other words, a user story describes a goal the user has when using the system. User stories are a basic concept in Agile development because they focus on simplicity, value added, and user collaboration. They document the functional requirements quickly and less formally than traditional requirements modeling by focusing on *who*, *what*, and *why* for each function. The users and stakeholders are responsible for identifying the user stories.

In meetings with stakeholders, analysts encourage participants to write out each user story on an index card or on a shared whiteboard app. The objective is to get a potential user to articulate what he or she wants to do with the new system. A standard template helps users think through what they want and why they want it. The standard template for a user story looks like this:

“As a <role played>, I want to <goal or desire> so that <reason or benefit>.”

For example, some user stories for a bank teller might be:

- “As a teller, I want to make a deposit to quickly serve more customers.”
- “As a teller, I want to balance the cash drawer to assure there were no errors.”

As a customer of the bank using an ATM machine, some user stories might be:

- “As a bank customer, I want to withdraw cash and feel confident the stack of cash I get is the correct amount.”
- “As a bank customer, I want to deposit a check and feel confident the deposit is recorded correctly.”

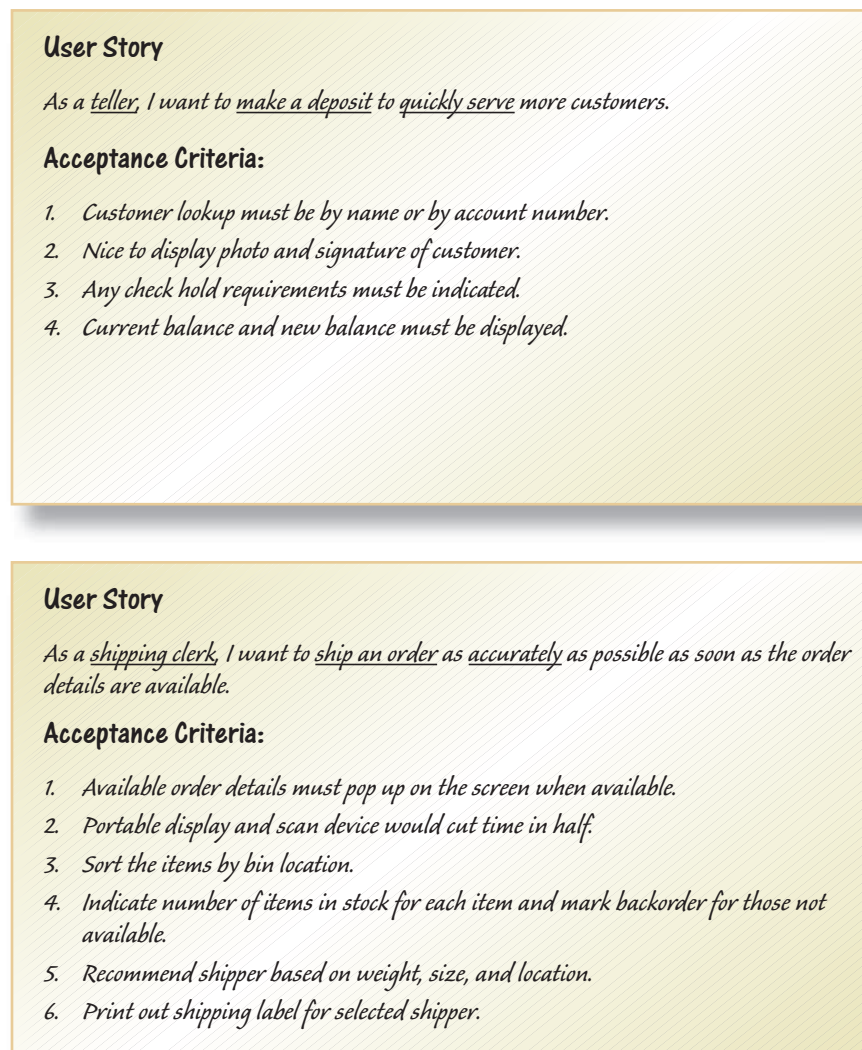
acceptance criteria features that must be present in the final system for the user to be satisfied

A final part of a user story is the **acceptance criteria**. These indicate the features that must be present for the user to be satisfied with the resulting implementation. They focus on functionality, not on features or user-interface design. For example, the following are the acceptance criteria for the user story “bank teller making a deposit”:

1. Customer lookup must be by name or by account number.
2. It would be nice to display photo and signature of customer.
3. Any check hold requirements must be indicated.
4. Current balance and new balance must be displayed.

The programmer analyst uses the acceptance criteria to clarify the expectations of the user and to verify the user is looking at the user story at an appropriate level of analysis. When the user story is implemented and refined, the acceptance criteria are used for testing. Some consider it much like a contract between the developers and the users that limits controversy later in the project. **Figure 3-1** shows two user stories handwritten on index cards. The first user story is for the bank teller example just discussed. The other user story is for a shipping clerk responsible for shipping the items on a new order for RMO.

FIGURE 3-1 Two user stories with acceptance criteria



use case an activity that the system performs in response to a request by a user

A **use case** is an activity the system performs in response to a request by a user. In Chapter 1, the RMO Tradeshow System example had a list of uses that included *Look up supplier*, *Enter/update product information*, and *Look up product information*. Two techniques are recommended for identifying use cases: the user goal technique and the event decomposition technique. Use case techniques place the responsibility for identifying and detailing the requirements on the system developers. The developers typically interview all types of users and stakeholders, and then make and refine notes about each use case. Some of the more complex use cases are modeled in more detail by the developers before turning the uses cases over to the programmer analysts for design and implementation.

■ Use Cases and the User Goal Technique

user goal technique a technique to identify use cases by determining what specific goals or objectives must be completed by the system for the user

“User stories will help analysts identify and define use cases, which are the primary focus of this chapter.”

One approach to identifying use cases, called the **user goal technique**, is to ask users to describe their goals for using the new or updated system. The analyst first identifies all the users, categorizes them by user type, and then conducts a structured interview with each user. By focusing on one type of user at a time, the analyst can systematically address the problem of identifying use cases.

During the interview, the analyst guides the user to identify specific ways the computer system could help the user perform his or her assigned tasks. The overarching objective is to identify what the system could do to improve the user’s performance and productivity. Subsidiary goals might include streamlining tasks the user currently performs, or enabling the user to perform new tasks that are not possible or practical with the current system. As these goals are uncovered and described, the analyst probes for specific requests from the user and desired responses from the proposed system, which the analyst documents as use cases. Although the users are the ultimate source of this information, they often require guidance from the analyst to think beyond the boundaries of the ways they currently approach their jobs.

Consider various user goals for the RMO Consolidated Sales and Marketing System (CSMS) introduced in Chapter 2. In an example like this, the analyst might talk to the people in the Shipping Department to identify their specific goals. These might include: *Ship items*, *Track shipment*, and *Create item return*. The Marketing Department might identify goals like *Add/update product information*, *Add/update promotion*, and *Produce sales history report*. When considering the goals of the prospective customer, the analyst might ask RMO users from different departments to think about the system from the customer’s viewpoint and to imagine the value-added features and functions that would make RMO appealing and useful. Additionally, focus groups of actual customers might be formed to pinpoint their wants and needs. Goals identified for potential customers might include *Search for item*, *Fill shopping cart*, and *View product rating and comments*. **Figure 3-2** lists a few of the user goals for potential users of the CSMS. Note that for the Shipping personnel, there is a use case named *Ship order*, which corresponds to the same user story identified in Figure 3-1.

The user goal technique for identifying use cases includes these steps:

1. Identify all the potential users for the new system.
2. Classify the potential users in terms of their functional role (e.g., shipping, marketing, sales).
3. Further classify potential users by organizational level (e.g., operational, management, executive).

FIGURE 3-2 Identifying use cases with the user goal technique

User	User goal and resulting use case
Potential customer	Search for item Fill shopping cart View product rating and comments
Marketing manager	Add/update product information Add/update promotion Produce sales history report
Shipping personnel	Ship order Track shipment Create item return

© Cengage Learning®

- Interview each type of user to determine the specific goals they will have when using the new system. Start with goals they currently have and then get them to imagine innovative functions they think would add value. Encourage them to state each goal in the imperative verb-noun form, such as *Add customer*, *Update order*, and *Produce month-end report*.
- Create a list of preliminary use cases organized by type of user.
- Look for duplicates with similar use case names and resolve inconsistencies.
- Identify where different types of users need the same use cases.
- Review the completed list with each type of user and then with interested stakeholders.

■ Use Cases and Event Decomposition

event decomposition technique a technique to identify use cases by determining the business events to which the system must respond

elementary business processes (EBP) the most fundamental task in a business process, which leaves the system and data in a quiescent state; usually performed by one person in response to a business event

event something that occurs at a specific time and place, can be precisely identified, and must be remembered by the system

The most comprehensive technique for identifying use cases is the event decomposition technique. The **event decomposition technique** begins by identifying all the business events the information system responds to, with each event leading to a use case. Starting with business events helps the analyst define each use case at the appropriate level of detail. For example, one analyst might identify a use case as typing in a customer name on a form. A second analyst might identify a use case as the entire process of adding a new customer. A third analyst might even define a use case as working with customers all day, which could include adding new customers, updating customer records, deleting customers, following up on late-paying customers, or contacting former customers. The first example is too narrow to be useful. Conversely, working with customers all day—the third example—is too broad to be useful. The second example defines a complete user goal, which is the right level of analysis for a use case.

The appropriate level of detail for identifying use cases is one that focuses on **elementary business processes (EBPs)**. An EBP is a task that is performed by one person in one place in response to a business event, adds measurable business value, and leaves the system and its data in a stable and consistent state. In Figure 3-2, the RMO CSMS potential customer use cases *Search for item*, *Fill shopping cart*, and *View product rating and comments* are good examples of elementary business processes. *Fill shopping cart* is a response to the business event “Customer wants to shop.” There is one person filling the cart, and there is measurable value for the customer as items are added to the cart. When the customer stops adding items and moves to another task, the system remembers the current cart and is ready to switch to the new task.

Note that each EBP (and thus each use case) occurs in response to a business event. An **event** occurs at a specific time and place, can be described, and should be remembered by the system. Events drive or trigger all processing that a system does, so listing events and analyzing them makes sense when you need to define system requirements by identifying use cases.

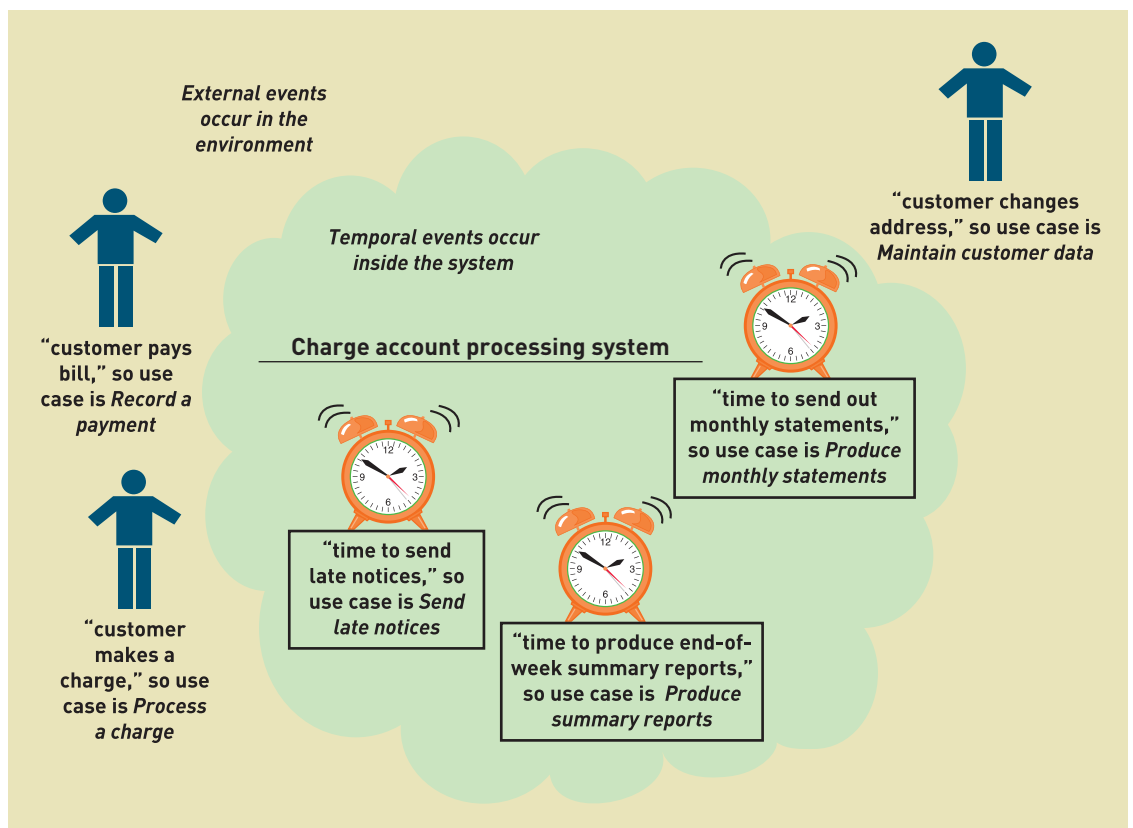
■ Event Decomposition Technique

When defining the requirements for a system, it is useful to begin by asking, “What business events occur that will require the system to respond?” By asking about the events that affect the system, you direct your attention to the external environment and look at the system as a black box. This means you don’t see the underlying functions, just the input and results. This initial perspective helps keep your focus on a high-level view of the system (looking at the scope), rather than on the inner workings of the system. It also focuses your attention on the system’s interfaces with outside people and other systems.

Some events that are important to a retail store’s charge account processing system are shown in **Figure 3-3**. The functional requirements are defined by use cases based on six events. A customer triggers three events: “customer pays a bill,” “customer makes a charge,” and “customer changes address.” The system responds with three use cases: *Record a payment*, *Process a charge*, or *Maintain customer data*. Three other events are triggered inside the system by reaching a point in time: “time to send out monthly statements,” “time to send late notices,” and “time to produce end-of-week summary reports.” The system responds with use cases that carry out what it is time to do: *Produce monthly statements*, *Send late notices*, and *Produce summary reports*. Describing this system in terms of events keeps the focus of the charge account system on the business requirements and the elementary business processes. The result is a list of use cases triggered by business events at the right level of analysis.

Using events to define functional requirements was first emphasized for real-time systems in the early 1980s. Real-time systems must react immediately

FIGURE 3-3 Events in a charge account processing system that lead to use cases



to events in the environment. Early real-time systems include manufacturing process control systems and avionics guidance systems. For example, in process control, if a vat of chemicals is full, then the system needs to *Turn off the fill valve*. The relevant event is “vat is full,” and the system needs to respond to that event immediately. In an airplane guidance system, if the plane’s altitude drops below 5,000 feet, then the system needs to *Turn on the low-altitude alarm*.

Most information systems now being developed are so interactive that they can be thought of as real-time systems. In fact, people expect a real-time response to almost everything. Thus, use cases for business systems are often identified by using the event decomposition technique.

■ Types of Events

There are three types of events to consider when using the event decomposition technique to identify use cases: external events, temporal events, and state events (also called *internal events*). The analyst begins by trying to identify and list as many of these events as possible, refining the list while talking with system users.

■ External Events

external event an event that occurs outside the system, usually initiated by an external agent

actor an external agent; a person, group or external system that interacts with the system by supplying or receiving data

An **external event** is an event that occurs outside the system—usually initiated by an external agent or actor. An external agent (or **actor**) is a person or organizational unit that supplies or receives data from the system. To identify the key external events, the analyst first tries to identify all the external agents that might want something from the system. A classic example of an external agent is a customer. The customer may want to place an order for one or more products. This event is of fundamental importance to an order-processing system, such as the one needed by Ridgeline Mountain Outfitters. But other events are associated with a customer. Sometimes, a customer wants to return an ordered product, or a customer needs to pay the invoice for an order. External events such as these define what the system needs to be able to do. They are events that lead to important transactions that the system must process.

When describing external events, it is important to name the event so the external agent is clearly defined. The description should also include the action that the external agent wants to pursue. Thus, the event “Customer places an order” describes the external agent (a customer) and the action that the customer wants to take (to place an order for some products) that directly affects the system.

Important external events can also result from the wants and needs of people or organizational units inside the company (e.g., management requests for information). A typical event in an order-processing system might be “Management wants to check order status.” Perhaps managers want to follow up on an order for a key customer; the system must routinely provide that information.

Another type of external event occurs when external entities provide new information that the system simply needs to store for later use. For example, a regular customer reports a change in address, phone, or employer. Usually, one event for each type of external agent can be described to handle updates to data, such as “Customer needs to update account information.” **Figure 3-4** provides a checklist to help in identifying external events.

FIGURE 3-4 External event checklist

External events to look for include:

- ✓ External agent wants something resulting in a transaction
- ✓ External agent wants some information
- ✓ Data changed and needs to be updated
- ✓ Management wants some information

FIGURE 3-5 *Temporal event checklist*

Temporal events to look for include:

- √ Internal outputs needed
 - √ Management reports (summary or exception)
 - √ Operational reports (detailed transactions)
 - √ Internal statements and documents (including payroll)
- √ External outputs needed
 - √ Statements, status reports, bills, reminders

© Cengage Learning®

temporal event an event that occurs as a result of reaching a point in time

■ Temporal Events

A second type of event is a **temporal event**—an event that occurs as a result of reaching a point in time. Many information systems produce outputs at defined intervals, such as payroll systems that produce a paycheck every two weeks (or each month). Sometimes, the outputs are reports that management wants to receive regularly, such as monthly or weekly performance or exception reports. These events are different from external events in that the system should automatically produce the required output without being told to do so. In other words, no external agent or actor is making demands, but the system is supposed to generate information or other outputs when they are needed.

The analyst begins identifying temporal events by asking about specific deadlines that the system must accommodate. What outputs are produced at that deadline? What other processing might be required at that deadline? In a payroll system, a temporal event might be named “Time to produce biweekly payroll.” The event defining the need for a monthly summary report might be named “Time to produce monthly sales summary report.” **Figure 3-5** provides a checklist to use in identifying temporal events.

Temporal events do not have to occur on a fixed date. They can occur after a defined period of time has elapsed. For example, a bill might be given to a customer when a sale has occurred. If the bill has not been paid within 15 days, the system might send a late notice. The temporal event “Time to send late notice” might be defined as a point 15 days after the billing date.

■ State Events

state event an event that occurs when something happens inside the system that triggers some process

A third type of event is a **state event**—an event that occurs when something happens inside the system that triggers the need for processing. State events are also called *internal events*. For example, if the sale of a product results in an adjustment to an inventory record, and the inventory in stock drops below a reorder point, it is necessary to reorder. The state event might be named “Reorder point reached.” Often, state events occur as a consequence of external events. Sometimes, they are similar to temporal events, except the point in time cannot be defined.

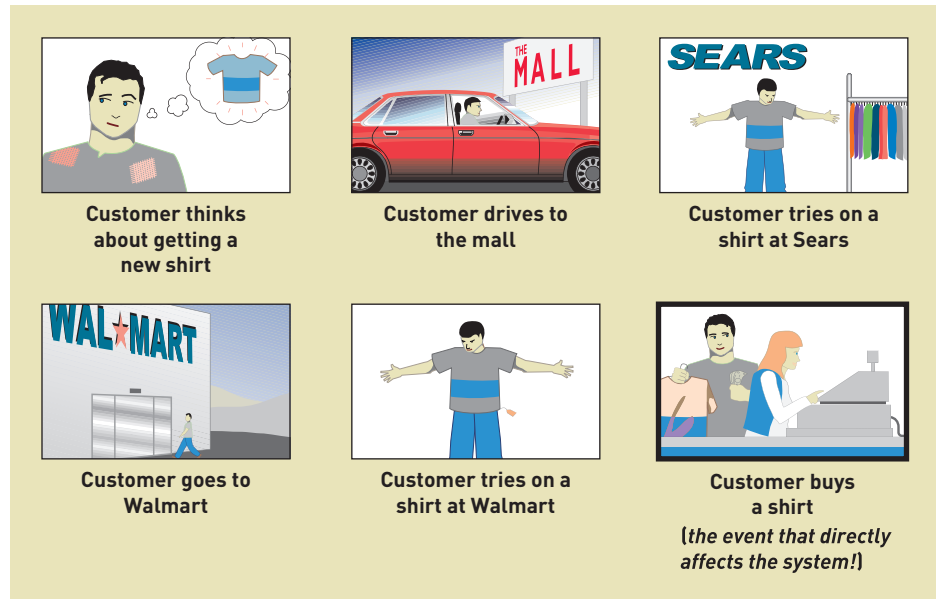
■ Identifying Events

Defining the events that affect a system is not always easy, but some guidelines can help an analyst think through the process.

■ Events Versus Prior Conditions and Responses

It is sometimes difficult to distinguish between an event and part of a sequence of prior conditions that leads up to the event. Consider a customer buying a shirt from a retail store (see **Figure 3-6**). From the customer’s perspective, this purchase involves a long sequence of events. The first event might be that the customer wants to get dressed. Then, the customer wants to wear a striped shirt. Next, the striped shirt appears to be worn out. The customer decides to drive to the mall, and he decides to go into Sears. Then, he tries on a striped shirt. At this point, the customer decides to leave Sears and go to Walmart to try on a

FIGURE 3-6 Sequence of actions that lead up to only one event affecting the system



shirt. Finally, the customer wants to purchase the shirt. The analyst has to think through such a sequence to arrive at the point where an event directly affects the system. In this case, the system is not affected until the customer is in the store, has a shirt in hand ready to purchase, and says “I want to buy this shirt.”

In other situations, it is not easy to distinguish between an external event and the system’s response. For example, when the customer buys the shirt, the system requests a credit card number and then the customer supplies the credit card. Is the act of supplying the credit card an event? In this case, no; it is part of the interaction that occurs while completing the original transaction.

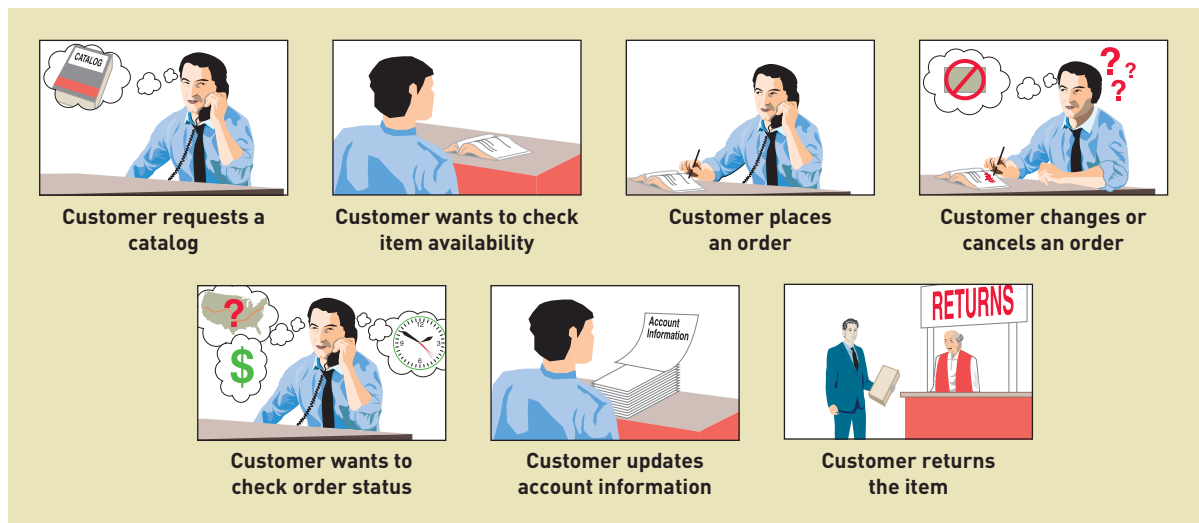
The way to determine whether an occurrence is an event or whether it is part of the interaction following the event is by asking if any long pauses or intervals occur (i.e., can the system transaction be completed without interruption?). Or is the system at rest again, waiting for the next transaction? After the customer wants to buy the shirt (the event), the process continues until the transaction is complete. There are no significant stops after the transaction begins. After the transaction is complete, the system is at rest, waiting for the next transaction to begin. The EBP concept defined earlier describes this as leaving the system and its data in a consistent state.

On the other hand, separate events occur when the customer buys the shirt by using his store credit card account. When the customer pays the bill at the end of the month, is the processing part of the interaction involving the purchase? In this case, no; the system records the transaction and then does other things. It does not halt all processes to wait for the payment. A separate event occurs later that results in sending the customer a bill. (This is a temporal event: “Time to send monthly bills.”) Eventually, another external event occurs (“Customer pays the bill”).

■ The Sequence of Events: Tracing a Transaction’s Life Cycle

A useful method for identifying events is to trace the sequence of events that might occur for a specific external agent or actor. In the case of Ridgeline Mountain Outfitters’ new CSMS, the analyst can think through all the possible transactions that might result from one new customer (see **Figure 3-7**). First, the customer wants a catalog or asks for some information about item availability, resulting in a name and address being added to the database. Then, the customer might want to place an order. Perhaps he or she will want to change the order—for example, correcting the size of the shirt or buying another shirt.

FIGURE 3-7 The sequence of “transactions” for one specific customer resulting in many events



Next, the customer might want to check the status of an order to find out the shipping date. Perhaps the customer has moved and wants an address change recorded for future catalog mailings. Finally, the customer might want to return an item. Thinking through this type of sequence can help identify events.

■ Technology-Dependent Events and System Controls

Sometimes, the analyst is concerned about events that are important to the system, but do not directly concern users or transactions. Such events typically involve design choices or system controls. During analysis, the analyst should temporarily ignore these events. However, they are important later for design.

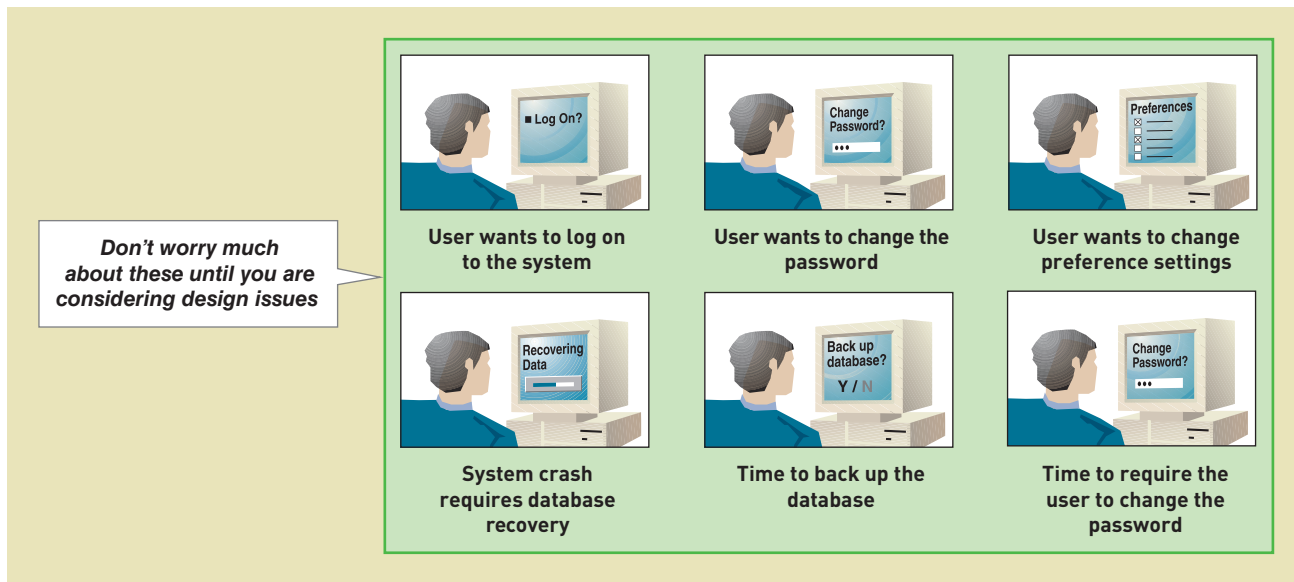
Some examples of events that affect design issues include external events that refer to the physical system, such as logging on. Although important to the final operation of the system, such implementation details should be deferred. At this stage, the analyst should focus only on the functional requirements (i.e., the work that the system needs to complete). A functional requirements model does not need to indicate how the system is actually implemented, so the model should omit the implementation details.

Most of these physical system events involve **system controls**, which are checks or safety procedures put in place to protect the integrity of the system. For example, logging on to a system is required because of system security controls. Other controls protect the integrity of the database, such as backing up the data every day. These controls are important to the system, and they will certainly be added to the system during design. But spending time on system controls during analysis only adds details to the requirements model that users are not typically very concerned about; they trust the system developers to take care of such details. One way to help decide which events apply to system controls is to assume that technology is perfect. The **perfect technology assumption** states that events should be included during analysis only if the system would be required to respond under perfect conditions (i.e., with equipment never breaking down, capacity for processing and storage being unlimited, and people operating the system being completely honest and never making mistakes). By pretending that technology is perfect, analysts can eliminate events like “Time to back up the database” because they can assume that the disk will never crash. Again, during design, the project team adds these controls because technology is obviously not perfect. **Figure 3-8** lists some examples of events that can be deferred until the developer is designing in system controls.

system controls checks or safety procedures to protect the integrity of the system and the data

perfect technology assumption the assumption that a system runs under perfect operating and technological conditions

FIGURE 3-8 Events deferred until designing system controls



■ Steps in the Event Decomposition Technique

To summarize, the event decomposition technique for identifying use cases includes these steps:

1. Consider the external events in the system environment that require a response from the system by using the checklist shown in Figure 3-4.
2. For each external event, identify and name the use case that the system requires.
3. Consider the temporal events that require a response from the system by using the checklist shown in Figure 3-5.
4. For each temporal event, identify and name the use case that the system requires and then establish the point of time that will trigger the use case.
5. Consider the state events that the system might respond to, particularly if it is a real-time system in which devices or internal state changes trigger use cases.
6. For each state event, identify and name the use case that the system requires and then define the state change.
7. When events and use cases are defined, check to see if they are required as part of analysis by using the perfect technology assumption. Do not include events that involve such system controls as login, logout, change password, and backup or restore the database, as these are put in as system controls.

■ Use Cases in the Ridgeline Mountain Outfitters Case

The RMO CSMS involves a variety of use cases, many of them just discussed. The analysts working on the new system have used all three techniques for identifying, validating, and refining use cases. The initial system vision (discussed in Chapter 2) identified four subsystems: the Sales subsystem, the Order Fulfillment subsystem, the Customer Account subsystem, and the Marketing subsystem. As work progressed, the analysts combined reports required by each subsystem into a fifth subsystem called the Reporting subsystem. In a system this size, the analyst should organize the use cases by subsystem to help track

FIGURE 3-9 Use cases and brief descriptions

Use case	Brief use case description
<i>Create customer account</i>	User/actor enters new customer account data, and the system assigns account number, creates a customer record, and creates an account record.
<i>Look up customer</i>	User/actor enters customer account number, and the system retrieves and displays customer and account data.
<i>Process account adjustment</i>	User/actor enters order number, and the system retrieves customer and order data; actor enters adjustment amount, and the system creates a transaction record for the adjustment.

© Cengage Learning®

which subsystem is responsible for each use case. The analyst should also identify which use cases involve more than one type of user.

It is important to recognize that this list of uses cases will continue to evolve as the project progresses. Additional use cases will be added, some might be eliminated, and some might be combined. It is helpful to immediately describe some of the details of each use case, preferably in one sentence. This brief description is usually expanded to record more of the details when the developers are designing and implementing the use case (see Chapter 5). Some examples of **brief use case descriptions** are shown in **Figure 3-9**. **Figures 3-10a** through **3-10e** show the initial list of use cases for the RMO CSMS along with the users. Note that many use cases have more than one user.

Sometimes, it is useful to create diagrams that visually depict use cases and how they are organized. The **use case diagram** is the UML model used to illustrate use cases and their relationship to users. Recall from Chapter 2 that Unified Modeling Language (UML) is the standard set of diagrams and model constructs used in system development. You saw an example of a use case diagram in Chapter 1. The notation is fairly simple.

■ Use Cases, Actors, and Use Case Diagram Notation

Implied in most use cases is a person who uses the system, whom we have referred to up to this point as the user. In UML, that person is called an *actor*. An actor is always outside the automation boundary of the system but may be part of the manual portion of the system. Sometimes, the actor for a use case is not a person; instead, it can be another system or device that receives services from the system.

Figure 3-11 shows the basic parts of a use case diagram. A simple stick figure represents an actor. The stick figure is given a name that characterizes the role the actor is playing. The use case itself is represented by an oval with the name of the use case inside. The connecting line between the actor and the use case indicates that the actor is involved with that use case. Finally, the **automation boundary**, which defines the border between the computerized portion of the application and the people operating the application, is shown as a rectangle containing the use case. The actor's communication with the use case crosses the automation boundary. The example in **Figure 3-11** shows the actor as a shipping clerk and the use case *Ship items*.

■ Use Case Diagram Examples

Figure 3-12 shows a more complete use case diagram for a subsystem of the RMO CSMS: the Customer Account subsystem. The information in **Figure 3-10c** is recast as a single use case diagram to visually highlight the uses cases and actors for an individual subsystem. In this example, the customer, customer service representative, and store sales representative are all allowed to access the system directly. As indicated by the relationship lines, each actor can access the use case *Create/update customer account*. The customer might do

brief use case description an often one-sentence description that provides a quick overview of a use case

use case diagram the UML model used to illustrate use cases and their relationships to actors

automation boundary the boundary between the computerized portion of the application and the users who operate the application but are part of the total system

FIGURE 3-10a Use cases and actors for CSMS Sales Subsystem

CSMS Sales Subsystem	
Use cases	Users/actors
Search for item	Customer, customer service representative, store sales representative
View product comments and ratings	Customer, customer service representative, store sales representative
View accessory combinations	Customer, customer service representative, store sales representative
Fill shopping cart	Customer
Empty shopping cart	Customer
Check out shopping cart	Customer
Fill reserve cart	Customer
Empty reserve cart	Customer
Convert reserve cart	Customer
Create phone sale	Customer service representative
Create store sale	Store sales representative

© Cengage Learning®

FIGURE 3-10b Use cases and actors for CSMS Order Fulfillment Subsystem

CSMS Order Fulfillment Subsystem	
Use cases	Users/actors
Ship items	Shipping
Manage shippers	Shipping
Create backorder	Shipping
Create item return	Shipping, customer
Look up order status	Shipping, customer, management
Track shipment	Shipping, customer, marketing
Rate and comment on product	Customer
Provide suggestion	Customer
Review suggestions	Management

© Cengage Learning®

FIGURE 3-10c Use cases and actors for CSMS Customer Account Subsystem

CSMS Customer Account Subsystem	
Use cases	Users/actors
Create/update customer account	Customer, customer service representative, store sales representative
Process account adjustment	Management
Send message	Customer
Browse messages	Customer
Request friend linkup	Customer
Reply to linkup request	Customer
Send/receive partner credits	Customer
View "mountain bucks"	Customer
Transfer "mountain bucks"	Customer

© Cengage Learning®

FIGURE 3-10d Use cases and actors for CSMS Marketing Subsystem

CSMS Marketing Subsystem	
Use cases	Users/actors
Add/update product information	Merchandising, marketing
Add/update promotion	Marketing
Add/update accessory package	Merchandising
Add/update business partner link	Marketing

© Cengage Learning®

FIGURE 3-10e Use cases and actors for CSMS Reporting Subsystem

CSMS Reporting Subsystem	
Use cases	Users/actors
Produce daily transaction summary report	Management
Produce sales history report	Management, marketing
Produce sales trends report	Marketing
Produce customer usage report	Marketing
Produce shipment history report	Management, shipping
Produce promotion impact report	Marketing
Produce promotional partner activity report	Management, marketing

© Cengage Learning®

FIGURE 3-11 A simple use case with an actor

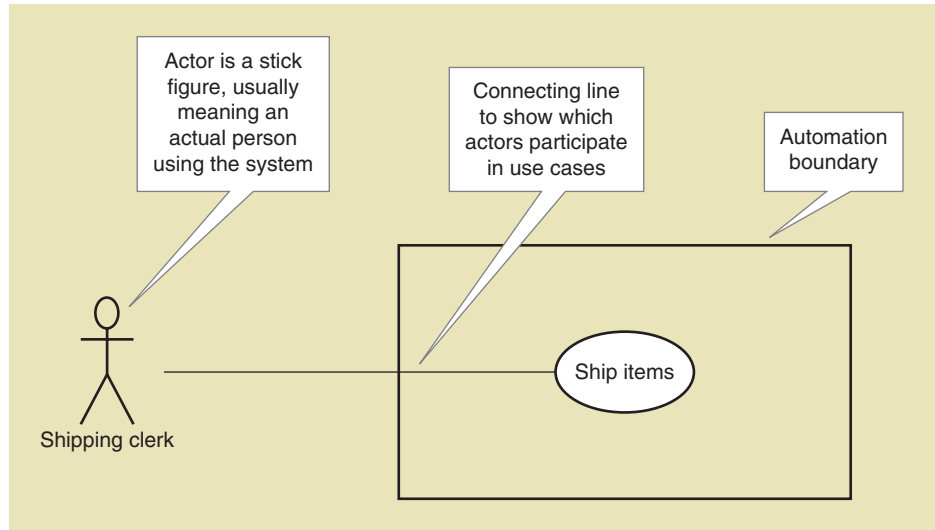


FIGURE 3-12 A use case diagram of the Customer Account subsystem for RMO, showing all actors

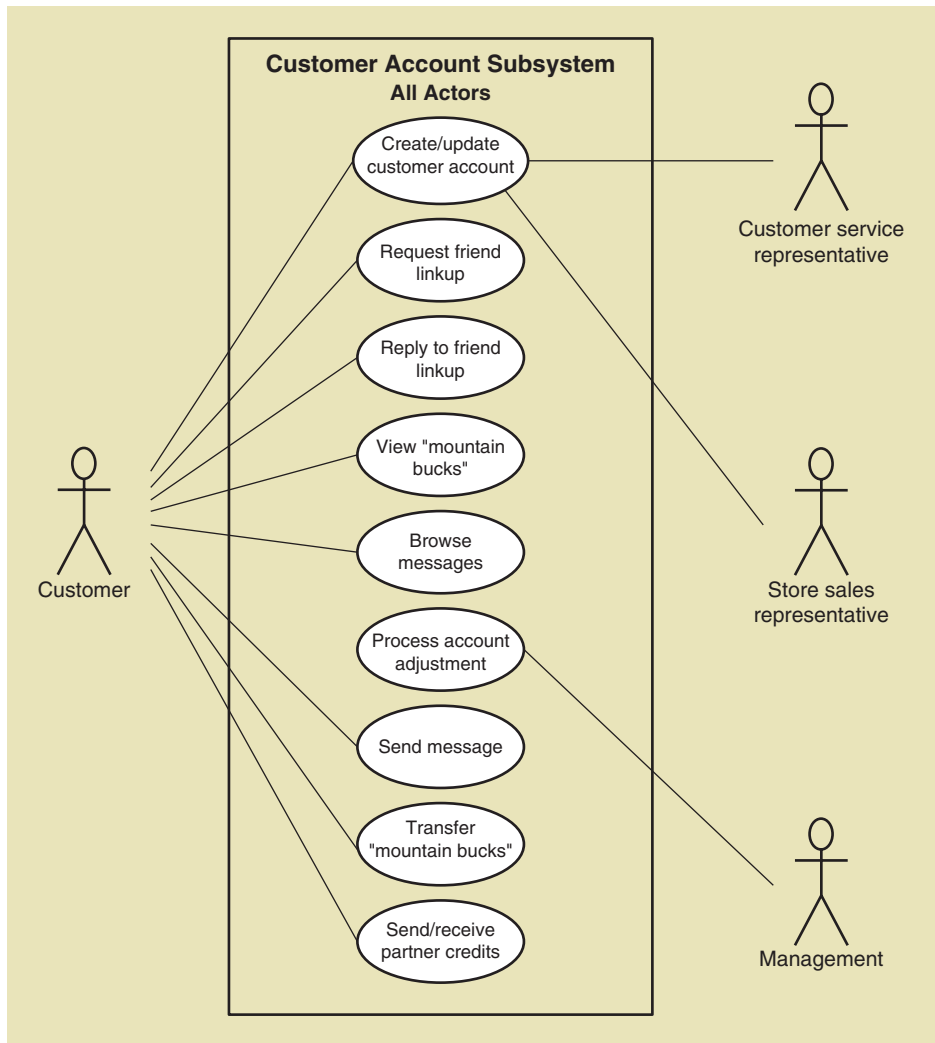
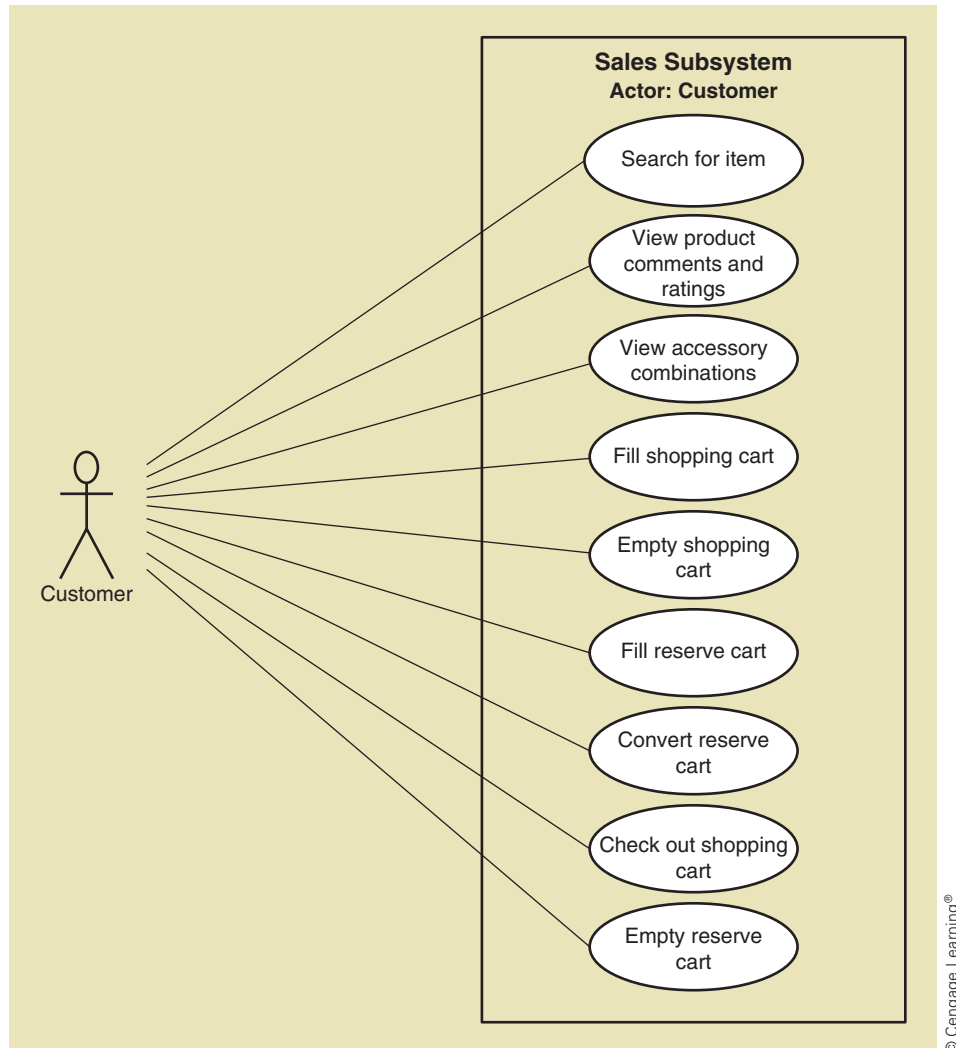


FIGURE 3-13 All use cases involving the customer actor for the Sales subsystem



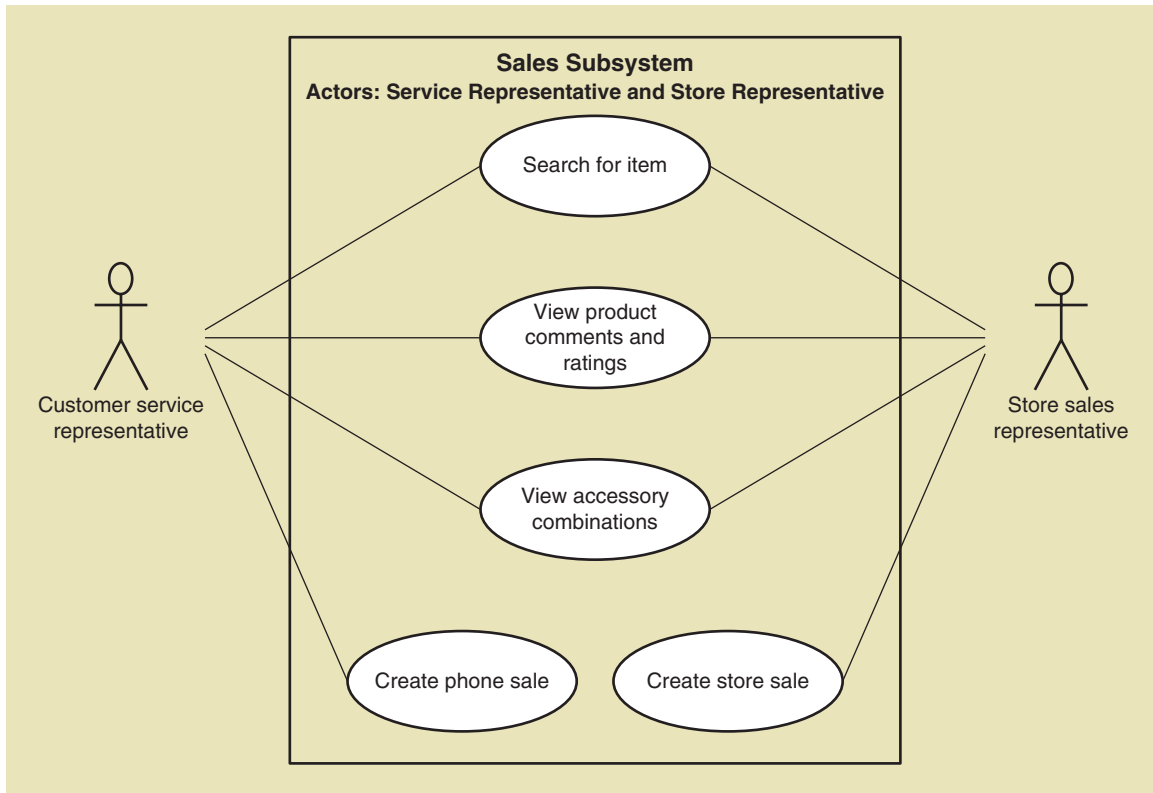
this when checking out online. The customer service representative might do this when talking to a customer on the phone. The store sales representative might do this when dealing with the customer in a store. Only a member of management can process an account adjustment. The other use cases are included only for the customer.

There are many ways to organize use case diagrams for communicating with users, stakeholders, and project team members. One way is to show all use cases invoked by a particular actor (i.e., from the user's viewpoint). This approach is often used during requirements definition because the systems analyst may be working with a particular user and identifying all the functions that user performs with the system. **Figure 3-13** illustrates this viewpoint, showing all the use cases involving the customer for the Sales subsystem. **Figure 3-14** shows use cases involving the customer service representative and the store sales representative for the Sales subsystem. Analysts can expand this approach to include all the use cases invoked by any department, regardless of the subsystem, or all use cases important to a specific stakeholder.

■ «includes» Relationships

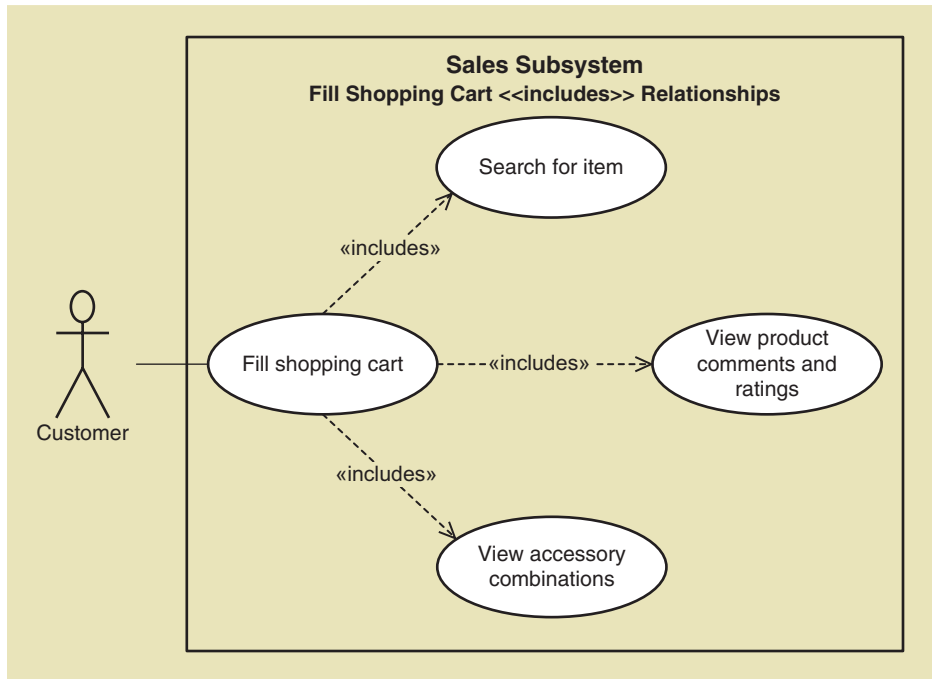
Frequently during the development of a use case diagram, it becomes apparent that one use case might use the services of another use case. For example, in the Sales subsystem use case diagram shown in **Figure 3-14**, the customer might search

FIGURE 3-14 Use cases involving the customer service representative and store sales representative for the Sales subsystem



© Cengage Learning®

FIGURE 3-15 A use case diagram of the Fill shopping cart «includes» relationships



© Cengage Learning®

for an item, view product comments and ratings, and view accessory combinations before beginning to fill the shopping cart. However, while filling the shopping cart, the customer might also search for an item, view product comments, and view accessories. Therefore, one use case uses, or “includes,” another use case. **Figure 3-15** shows a use case diagram emphasizing this aspect of use cases.

«includes» relationship a relationship between use cases in which one use case is stereotypically included within the other use case

Fill shopping cart also includes *Search for item*, *View product comments and ratings*, and *View accessory combinations*. Thus, the Customer can view comments initially, and also while carrying out the *Fill shopping cart* use case. The relationship between these use cases is denoted by the dashed connecting line with the arrow that points to the use case that is included. The relationship is read *Fill shopping cart includes Search for item*. Sometimes, this relationship is referred to as the **«includes» relationship** or the «uses» relationship. Note that the word *includes* is enclosed within guillemets in the diagram; this is the way to refer to a stereotype in UML. It means that the relationship between one use case and another use case is a stereotypical «includes» relationship.

■ Developing a Use Case Diagram

Analysts create a variety of use case diagrams to communicate with users, stakeholders, management, and team members. The steps to develop use case diagrams are:

1. Identify all the stakeholders and users who would benefit by having a use case diagram.
2. Determine what each stakeholder or user needs to review in a use case diagram. Typically, a use case diagram might be produced for each subsystem, for each type of user, for use cases with the «includes» relationship, and for use cases that are of interest to specific stakeholders.
3. For each potential communication need, select the use cases and actors to show and draw the use case diagram. There are many software packages that can be used to draw use case diagrams.
4. Carefully name each use case diagram and then note how and when the diagram should be used to review use cases with stakeholders and users.

CHAPTER SUMMARY

This chapter is the first of three chapters that present techniques for modeling a system's functional requirements. A key early step in the modeling process is to identify and list the user stories or use cases that define the functional requirements for the system. User stories are written by end users and stakeholders during requirements meetings with the developers. They are used with most highly Agile development methodologies. Use cases are very similar to user stories, but they are often modeled in more detail than user stories and are the responsibility of the developers. Use cases can be identified by using the user goal technique and the event decomposition technique. The user goal technique initially identifies types of system end users, called *actors*. Then, users are asked to list specific user goals they have when using the system to support their work. The event decomposition technique initially identifies events that require a response from the

system. An event is something that can be described, something that occurs at a specific time and place, and something worth remembering. External events occur outside the system—usually triggered by someone who interacts with the system. Temporal events occur at a defined point in time, such as the end of a work day or the end of every month. State or internal events occur based on an internal system change. For each event, a use case is identified and named. The event decomposition technique helps ensure that each use case is identified at the elementary business process (EBP) level of detail. Each use case identified by the analyst is further documented by a brief use case description and by identifying the actors. UML use case diagrams are drawn to document use cases and their actors. Many different use case diagrams are drawn based on the need to review use cases with various stakeholders, users, and team members.

KEY TERMS

acceptance criteria	event	system controls
actor	event decomposition technique	temporal event
automation boundary	external event	use case
brief use case description	«includes» relationship	use case diagram
elementary business process (EBP)	perfect technology assumption	user goal technique
	state event	user story

REVIEW QUESTIONS

1. What are the five activities of systems analysis, and which activity is discussed beginning with this chapter?
2. What is a user story? What is a use case?
3. What are the two techniques used to identify use cases?
4. Describe the user goal technique for identifying use cases.
5. What are some examples of users with different functional roles and at different operational levels?
6. What are some examples of use case names that correspond to your goals as a student going through the college registration process? Be sure to use the verb-noun naming convention.
7. What is the overarching objective of asking users about their specific goals?
8. How many types of users can have the same user goals for using the system?
9. Describe the event decomposition technique for identifying use cases.
10. Why is the event decomposition technique considered more comprehensive than the user goal technique?
11. What is an elementary business process (EBP)?
12. Explain how the event decomposition technique helps identify use cases at the right level of analysis.
13. What is an event?
14. What are the three types of events?
15. Define an external event and give an example that applies to a checking account system.
16. Define a temporal event and give an example that applies to a checking account system.
17. What are system controls, and why are they not considered part of the users' functional requirements?
18. What is the perfect technology assumption?
19. What are three examples of events that involve system controls that should not be included initially because of the perfect technology assumption?
20. What is a brief use case description?
21. What is UML?
22. What is the purpose of UML use case diagrams?
23. What is another name for "actor" in UML, and how is it represented on a use case diagram?
24. What is the automation boundary on a use case diagram, and how is it represented?
25. How many actors can be related to a use case on a use case diagram?
26. Why might a systems analyst draw many different use case diagrams when reviewing use cases with end users?
27. What is the «includes» relationship between two use cases?

PROBLEMS AND EXERCISES

1. Consider the situation where a student organization is exploring its requirements for a system that manages its membership and finances. Based on what you know about student organizations, write user stories using the standard template for the following potential users: membership director, finance director, faculty advisor. Add acceptance criteria for each user story based on how you imagine the system might work.

2. Review the external event checklist in Figure 3-4 and then think about a university course registration system. What is an example of an event of each type in the checklist? Name each event by using the guidelines for naming an external event.
3. Review the temporal event checklist in Figure 3-5. Would a student grade report be an internal or external output? Would a class list for the instructor be an internal or external output? What are some other internal and external outputs for a course registration system? Using the guidelines for naming temporal events, what would you name the events that trigger these outputs?
4. Consider the following sequence of actions taken by a customer at a bank. Which action is the event the analyst should define for a bank account transaction processing system? (1) Kevin gets a check from Grandma for his birthday. (2) Kevin wants a car. (3) Kevin decides to save his money. (4) Kevin goes to the bank. (5) Kevin waits in line. (6) Kevin makes a deposit in his savings account. (7) Kevin grabs the deposit receipt. (8) Kevin asks for a brochure on auto loans.
5. Consider the perfect technology assumption, which states that use cases should be included during analysis only if the system would be required to respond under perfect conditions. Could any of the use cases listed for the RMO CSMS be eliminated based on this assumption? Explain. Why are such use cases as *Log on to the system* and *Back up the database* required only under imperfect conditions?
6. Visit some Web sites of car manufacturers, such as Honda, BMW, Toyota, and Acura. Many of these sites have a use case that is typically named *Build and price a car*. As a potential customer, you can select a car model, select features and options, and get the car's suggested price and list of specifications. Write a brief use case description for this use case (see Figure 3-9).
7. Again looking at a Web site for one of the car manufacturers, consider yourself a potential buyer and then identify all the use cases included on the site that correspond to your goals.
8. Set up a meeting with a librarian. During your meeting, ask the librarian to describe the situations that the book checkout system needs to respond to. List these external events. Now ask about points in time, or deadlines, that require the system to produce a statement, notice, report, or other output. List these temporal events. Does it seem natural for the librarian to describe the system in this way? List each event and then name the resulting use case.
9. Again considering the library, ask some students what their goals are in using the library system. Also ask some library employees about their goals in using the system. Name these goals as use cases (verb-noun) and discuss whether student users have different goals than employee users.
10. Visit a restaurant or the college food service to talk to a server (or talk with a friend who is a food server). Ask about the external events and temporal events, as you did in exercise 8. What are the events and resulting use cases for order processing at a restaurant?
11. Review the procedures for course registration at your university and then talk with the staff in advising, in registration, and in your major department. Think about the sequence that goes on over an entire semester. What are the events that students trigger? What are the events that your own department triggers? What are the temporal events that result in information going to students? What are the temporal events that result in information going to instructors or departments? List all the events and the resulting use cases that should be included in the system.
12. Refer to the RMO CSMS Order Fulfillment subsystem shown in Figure 3-10. Draw a use case diagram that shows all actors and all use cases. Use a drawing tool such as Microsoft Visio if it is available.
13. Again for the Order Fulfillment subsystem, draw a use case diagram showing just the use cases for the Shipping Department in preparation for a meeting with them about the system requirements. Use a drawing tool such as Microsoft Visio if it is available.
14. Refer to the RMO CSMS Marketing subsystem shown in Figure 3-10. Draw a use case diagram that shows all actors and all use cases. Use a drawing tool such as Microsoft Visio if it is available.
15. Refer to the RMO CSMS Reporting subsystem shown in Figure 3-10. These reports were identified by asking users about temporal events, meaning points in time that require the system to produce information of value. In most actual systems today, an actor is assigned responsibility for producing the reports or other outputs when they are due. Recall that the actor is part of the system—the manual, nonautomated part. Thus, this is one way the “system” can be responsible for producing an output at a point in time. In the future, more outputs will be produced automatically. Draw a use case diagram that shows the use cases and actors, as shown in Figure 3-10. Use a drawing tool such as Microsoft Visio if it is available.

CASE STUDY

The State Patrol Ticket-Processing System

The purpose of the State Patrol ticket-processing system is to record moving violations, keep records of the fines paid by drivers when they plead guilty or are found guilty of moving violations, and notify the court that a warrant for arrest should be issued when such fines are not paid in a timely manner. A separate State Patrol system records accidents and the verification of financial responsibility (insurance). But a third system uses ticket and accident records to produce driving record reports for insurance companies. Finally, a fourth system issues, renews, or suspends driver's licenses. These four systems are obviously integrated, in that they share access to the same database; otherwise, they are operated separately by different departments of the State Patrol.

When an officer gives a ticket to a driver, a copy of the ticket is turned in and entered into the system. A new ticket record is created, and relationships to the correct driver, officer, and court are established in the database. If the driver pleads guilty, he or she mails in the fine in a preprinted envelope with the ticket number on it. In some cases, the driver claims innocence and wants a court date. When the envelope is returned without a check and the trial request box has an "X" in it, the system does the following: notes the plea on the ticket record; looks up driver, ticket, and officer information; and sends a ticket details report to the appropriate court. A trial date questionnaire form is also produced at the same time and is mailed to the driver. The instructions on the questionnaire tell the driver to fill in convenient dates and mail the questionnaire directly to the court. Upon receiving this information, the court schedules a trial date and notifies the driver of the date and time.

When the trial is completed, the court sends the verdict to the ticketing system. The verdict and trial date are recorded for the ticket. If the verdict is innocent, the system that produces driving record reports for insurance companies will ignore the ticket. If the verdict is guilty, the court gives the driver another envelope with the ticket number on it for mailing in the fine.

If the driver fails to pay the fine within the required period, the ticket-processing system produces a warrant request notice and sends it to the court. This happens if the driver does not return the original envelope within two weeks, or does not return the court-supplied envelope within two weeks of the trial date. What happens next is in the hands of the court. Sometimes, the court requests that the driver's license be suspended, and the system that processes driver's licenses handles the suspension.

1. To what events must the ticket-processing system respond? List each event, the type of event, the resulting use case, and the actor(s). Think carefully about who the actors are. Does the officer directly enter the ticket into the system? Or does a state patrol clerk do it when the paper ticket is received from the officer? The existing system uses paper forms, so for now assume the state patrol clerk enters all of the information. A new system would use the same use cases, but opportunities for efficiency and accuracy would lead to changes in who the actors are.
2. Write a brief use case description for each use case.
3. Draw a use case diagram for the ticket-processing system assuming the actors are based on your answers in question 1 (the existing system as described).

RUNNING CASE STUDIES

Community Board of Realtors®

One of the functions of the Board of Realtors introduced in Chapter 2 is to provide a Multiple Listing Service (MLS) system that supplies information that local real estate agents use to help them sell houses to their customers. Agents list houses for sale (listings) by contracting with homeowners. The agent works for a real estate office, which sends information on the

listing to the MLS. Therefore, any agent in the community can get information on the listing.

The MLS systems include lots of information on a listing. It is now common to include dozens of photos, video tours, Google map information, prior sales, and so forth. For now, let's keep it simple and assume a listing includes the address, year built, square feet,

number of bedrooms, number of bathrooms, owner name, owner phone number, asking price, and status code. An agent will directly request information from the MLS system on listings that match customer requirements. Information is provided on the house, on the agent who listed the house, and on the real estate office for which the agent works. This information is needed because an agent might want to call the listing agent to ask additional questions or call the homeowner directly to make an appointment to show the house. Although it seems dated, once each week, the MLS produces a listing booklet that contains information on all listings. These booklets are sent to some real estate agents because many think they are easier to flip

through and write on. Sometimes, agents and owners decide to change information about a listing, such as reducing the price, correcting previous information on the house, or indicating that the house is sold. The agent updates the information directly on the MLS system.

1. To what events must the MLS system respond based on the description above? List each event, the type of event, and the resulting use case. Be sure to consider all the use cases that would be needed to maintain the data in the MLS system.
2. Draw a use case diagram based on the use cases you identified in question 1. Include appropriate actors based on the case description.

The Spring Breaks ‘R’ Us Travel Service

Spring Breaks ‘R’ Us (SBRU), introduced in Chapter 2, includes many use cases that make up the functional requirements. Consider the following description of the Booking subsystem. A few weeks before Thanksgiving break, it is time to open the system to new bookings. Students usually want to browse through the resorts and do some planning. When a student or group of students wants to book a trip, the system allows it. Sometimes, a student needs to be added or dropped from the group or a group changes size and needs a different type of room. One month before the actual trip, it is time for the system to send out final payment requirement notices. Students cancel the booking or they pay their final bills. Students often want to look up their booking status and check on resort details. When they arrive at the resort, they need to check in; and when they leave, they need to check out.

1. Using the event decomposition technique for each event you identify in the description here, name the event, state the type of event, and name the resulting use case. Draw a use case diagram for these use cases.
2. Consider the new Social Networking subsystem that SBRU is researching that was described in Chapter 2. Think in terms of the user goal technique to identify as many use cases as you can think of that you would like to have in the system. SBRU is guessing you might want to join, send messages, and so forth, but there must be many interesting and useful things the system could do before, during, and after the trip. Draw a use case diagram for these use cases.

On the Spot Courier Services

Recall the On the Spot courier service introduced in Chapter 2. The details of the package pickup and delivery process are described here.

When Bill got an order, only on his phone at first, he recorded when he received the call and when the shipment would be ready for pickup. Sometimes, customers wanted immediate pickup; sometimes, they were calling to schedule a later time in the day for pickup.

Once he arrived at the pickup location, Bill collected the packages. It was not uncommon for the customer to have several packages for delivery. In addition to the name and address of the delivery location, he also recorded the time of pickup. He noted the desired delivery time, the location of the delivery, and the weight of the package to determine the courier cost. When he picked up the package, he

printed out a label with his portable printer that he kept in the delivery van.

At first, Bill required customers to pay at the time of pickup, but he soon discovered that there were some regular customers who preferred to receive a monthly bill for all their shipments. He wanted to be able to accommodate those customers. Bills were due and payable upon receipt.

To help keep track of all the packages, Bill decided that he needed to scan each package as it was sorted in the warehouse. This would enable him to keep good control of his packages and avoid loss or delays.

The delivery of a package was fairly simple. Upon delivery, he would record information about when the delivery was made and who received it. Because some of the packages were valuable, it was necessary in

those instances to have someone sign for the package upon delivery.

1. From this description as well as the information from Chapter 2, identify all the actors who will be using the system.
2. Using the actors who you identified in question 1, develop a list of use cases based on the user goal

Sandia Medical Devices

Recall the Sandia Medical Devices Real-Time Glucose Monitoring (RTGM) system introduced in Chapter 2. As the project began, interviews with patients and physicians about potential RTGM capabilities and interaction modes identified several areas of concern that will need to be incorporated into the system requirements and design. The relevant patient concerns include:

- **Viewing and interpreting data and trends.** Patients want to be able to view more than their current glucose level. They would like to see glucose levels over various time periods, with a specific focus on time periods during which their glucose was within and outside of acceptable ranges. A graphical view of the data is preferred, although some patients also want to be able to see actual numbers.
- **Entering additional data.** Some patients want to be able to enter text notes or voice messages to supplement glucose level data. For example, patients who see a high glucose alert might record voice messages describing how they feel or what they had recently eaten. Some patients thought that sharing such information with their health-care providers might be valuable, but others only wanted such information for themselves.

technique. Draw a use case diagram for these use cases.

3. Using the event decomposition technique for each event you identify in the description here, name the event, state the type of event, and name the resulting use case. Draw a use case diagram for these use cases.

Physicians expressed these concerns:

- They do not want to be the “first line of response” to all alerts. They prefer that nurses or physician assistants be charged with that role and that physicians be notified only when frontline personnel determine that an emergency situation exists.
- They want to be able to monitor and view past patient data and trends in much the same way as described for patients.
- They want all their actions to be logged and for patient-specific responses to be stored as part of the patient’s electronic medical record.

Perform the following tasks by using the information here as well as the system description in Chapter 2:

1. Identify all the actors who will use RTGM.
2. Using the actors who you identified in question 1, develop a list of use cases based on the user goal technique. Draw a use case diagram for these use cases.
3. Using the event decomposition technique for each event you identified in the description, name the event, state the type of event, and name the resulting use case. Draw a use case diagram for these use cases.

FURTHER RESOURCES

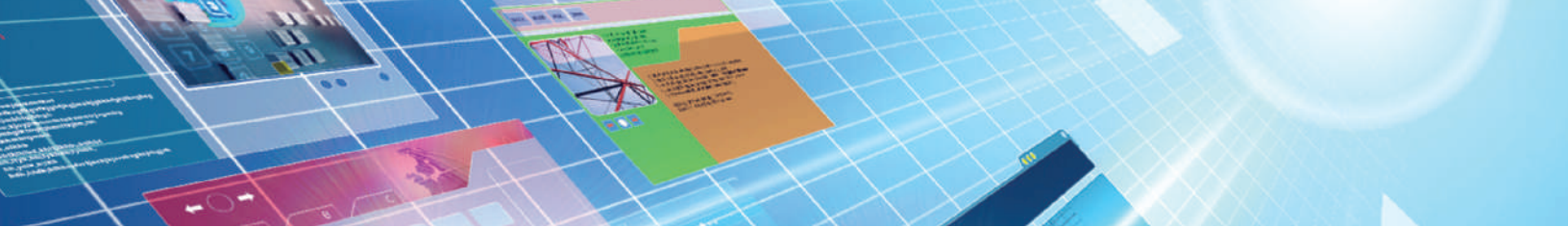
Classic and more recent texts include the following:

- Grady Booch, Ivar Jacobson, and James Rumbaugh, *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.
- Mike Cohn, *User Stories Applied*. Addison-Wesley, 2004.

Craig Larman, *Applying UML and Patterns* (3rd ed.). Prentice Hall, 2005.

Stephen McMenamin and John Palmer, *Essential Systems Analysis*. Prentice Hall, 1984.

Ed Yourdon, *Modern Structured Analysis*. Prentice Hall, 1989.



CHAPTER FOUR

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- ▶ Explain how the concept of “things” in the problem domain also defines requirements
- ▶ Identify and analyze data entities and domain classes needed in the system
- ▶ Read, interpret, and create an entity-relationship diagram
- ▶ Read, interpret, and create a domain model class diagram
- ▶ Understand and interpret the domain model class diagrams for RMO
- ▶ Read, interpret, and create a state machine diagram that models object behavior

CHAPTER OUTLINE

- ▶ “Things” in the Problem Domain
- ▶ The Entity-Relationship Diagram
- ▶ The Domain Model Class Diagram
- ▶ The State Machine Diagram—Identifying Object Behavior

OPENING CASE WAITERS ON CALL MEAL-DELIVERY SYSTEM (PART 2)

Recall that Waiters on Call has been working with Sam Wells on the requirements for its meal-delivery system. Sue and Tom Bickford want a new system that will automate and improve their specialty business of providing customer-ordered, home-delivered meals prepared by a variety of local restaurants. Sam did a great job of identifying the use cases required for the delivery service, which impressed the Bickfords. And while working on the use cases, he continued to note all the business terms and concepts that the Bickfords used as they described their operations. He followed up with questions about the types of things they work with each day, which they answered.

“Based on what you’ve told me,” Sam said, “I assume you will need the system to store information about the following types of things, which we call data entities or domain classes: restaurants, menu items, customers, and orders. I also think you’re

going to need to store information about the following types of things: drivers, addresses, routes, and order payments.”

The Bickfords readily agreed and added that it was important to know what route a restaurant was on and how far it might be to the customer’s address. They wanted drivers to be assigned to a route based on the distances from place to place.

“Yes, we need to decide how things need to be associated in the system,” Sam agreed. “Can you tell me if drivers pick up orders from several restaurants when they go out? Can you tell me how many items are usually included in one order? Do you note pickup times and delivery times? Do you need to plan the route so that hot dishes are delivered first?”

The Bickfords were further reassured that they had picked an analyst who was aware of the needs of their business.

■ Overview

Chapter 3 focused on identifying user stories or use cases to define the functional requirements for an information system. This chapter focuses on another key concept that defines requirements: things in the problem domain of system users. You first learned about these as domain classes in Chapter 1. You might have learned about these as data entities when studying database management, as they define the sources for the tables used in a relational database management system. Nearly all approaches to system development include identifying and modeling domain classes or data entities as an important task in the analysis activity *Define functional requirements*.

■ “Things” in the Problem Domain

problem domain the specific area (domain) of the user’s business need that is within the scope of the new system

Domain classes or data entities are what end users deal with when they do their work—for example, products, sales, shippers, shipments, and customers. These are often referred to as “things” in the context of a system’s problem domain. The **problem domain** is the specific area of the user’s business that is included within the scope of the new system. The new system involves working with and remembering these “things.” For example, some information systems need to store information about customers and products, so it is important for the analyst to identify all the important information about them. Often, things are related to the people who interact with the system or to other stakeholders. For example, a customer is a person who places an order, but the system needs to store information about that customer, so a customer is also a thing in the problem domain.

There are many techniques for identifying the important things in the problem domain that the system needs to remember. Two of them are introduced in this chapter: the brainstorming technique and the noun technique.

■ The Brainstorming Technique

brainstorming technique a technique used to identify problem domain classes in which developers work with users to identify classes by thinking about different types of things they use in their work

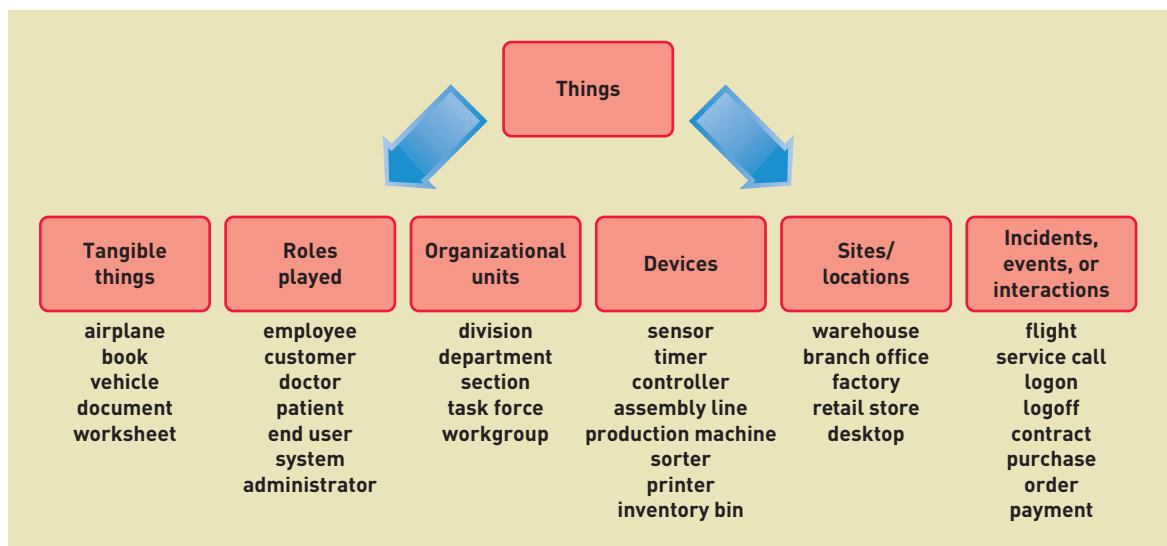
As with use cases, an analyst should ask the users to discuss the types of things they work with routinely. The analyst can ask about several types of things to help identify them. Different types of things are important to different users, so it is important to involve all types of users to help identify problem domain things. The **brainstorming technique** is used to identify problem domain classes where developers work with users to identify different types of things that they use in their work.

Figure 4-1 shows some types of things to consider that can be used to systematically help users do brainstorming. Tangible things are often the most obvious, such as an airplane, a book, or a vehicle. In the Ridgeline Mountain Outfitters case, a product in the warehouse and a vehicle in the fleet are tangible things of importance. Another common type of thing in an information system is a role played by a person, such as an employee, a customer, a doctor, or a patient. The role of customer is obviously a very important one in the Ridgeline Mountain Outfitters case. Many things in the problem domain can fit into more than one type. For example, a vehicle might be thought of as a device and also a tangible thing. Either way, the important point is to identify potential things in the problem domain.

Other types of things can include organizational units, such as a division, department, or workgroup. Similarly, a site or location, such as a warehouse, a store, or a branch office, might be an important thing in a system. Finally, information about an incident or an interaction can be a thing—information about an order, a service call, a contract, or an airplane flight. A sale, a shipment, and a return are all important incidents in the RMO case. Sometimes, these incidents are thought of as associations between things. For example, a sale is an association between a customer and an item of inventory. Initially, the analyst might simply list all these as things and then make adjustments as required by different approaches to analysis and design.

The analyst identifies these types of things by thinking about each use case, talking to users, and asking questions. For example, for each use case, what types of things are affected that the system needs to know about and store information about? The types of things shown in Figure 4-1 can be used to systematically brainstorm about what types of things might be involved in each use case. When a customer wants to buy from the Web site, the system needs to store information about the customer, the items ordered, the details about the sale itself—such as the date and payment terms—and the location of the items to

FIGURE 4-1 Types of things to use for the brainstorming technique



be shipped. For that one use case, the analyst can define tangible things (items ordered), roles played (customer), incidents or events (the sale), sites/locations (warehouse), and organizational units (shipping).

Here are the steps to follow when using the brainstorming technique:

1. Identify a user and a set of use cases or user stories.
2. Brainstorm with the user to identify things involved when carrying out the use case—that is, things about which information should be captured by the system.
3. Use the types of things (categories) to systematically ask questions about potential things, such as the following: Are there any tangible things you store information about? Are there any locations involved? Are there roles played by people that you need to remember?
4. Continue to work with all types of users and stakeholders to expand the brainstorming list.
5. Merge the results, eliminate any duplicates, and compile an initial list.

■ The Noun Technique

Another useful procedure for identifying things in the problem domain is called the **noun technique**. Recall that a noun is a person, place, or thing. Therefore, identifying nouns might help you identify what needs to be stored by the system. Begin by listing all the nouns that users mention when talking about the system. Nouns used to describe events, use cases, user stories, and the actors are potential things. Next, add to the list any additional nouns that appear in information about the existing system or that come up in discussions with stakeholders about the problem domain of the system. The list of nouns will become quite long, so the list will need to be trimmed and refined. How the noun technique differs from the brainstorming technique is that the analyst lists all nouns without thinking too much about them and without talking about them with users. Only later will the list be refined based on consultation with stakeholders and users.

Here are the steps to follow when using the noun technique:

1. Using the use cases, actors, and other information about the system—including inputs and outputs—identify all nouns. For the RMO CSMS, the nouns might include the following: customer, product item, sale, confirmation, transaction, shipping, bank, change request, summary report, management, transaction report, accounting, back order, back-order notification, return, return confirmation, fulfillment reports, prospective customer, marketing, customer account, promotional materials, charge adjustment, sale details, merchandising, and customer activity reports.
2. Using other information from existing systems, current procedures, and current reports or forms, add items or categories of information needed. For the RMO CSMS, these might include more detailed information, such as price, size, color, style, season, inventory quantity, payment method, and shipping address. Some of these items might be additional things, and some might be specific information (called attributes) about things you have already identified. Refine the list and then record assumptions or issues to explore.
3. As this list of nouns builds, you will need to refine it. Ask these questions about each noun to help you decide whether you should include it:

- Is it a unique thing the system needs to know about?
- Is it inside the scope of the system I am working on?
- Does the system need to remember more than one of these items?

Ask these questions about each noun to decide whether you should exclude it:

- Is it really a synonym for some other thing I have identified?
- Is it really just an output of the system produced from other information I have identified?

noun technique a technique used to identify things in the problem domain by finding and classifying the nouns in a dialog or description

- Is it really just an input that results in recording some other information I have identified?

Ask these questions about each noun to decide whether you should research it:

- Is it likely to be a specific piece of information (attribute) about some other thing I have identified?
 - Is it something I might need if assumptions change?
4. Create a master list of all nouns identified and then note whether each one should be included, excluded, or researched further.
 5. Review the list with users, stakeholders, and team members and then refine the list of things in the problem domain.

Figure 4-2 lists some of the nouns from the RMO CSMS, with notes about each one. As with the brainstorming technique, the initial list developed from this table is just a start. Much more work is needed to refine the list and define more information about each item in the list.

■ Attributes of Things

The noun technique involves listing all the nouns that come up in discussions or documents about the requirements. The list can become quite long because many of these nouns are actually attributes. Most information systems store and use specific pieces of information about each “thing” of interest, as shown for some nouns in Figure 4-2. The specific pieces of information are called **attributes**. For example, a customer has a name, a phone number, a credit limit, and so on. Each of these details is an attribute. So as you refine your list of nouns, you may redefine many nouns as attributes rather than fundamental “things.”

The analyst needs to identify all of the attributes of each thing that the system needs to store even if they don’t turn up on the initial list of nouns. One

attributes descriptive pieces of information about things or objects

FIGURE 4-2 Partial list of “things” based on nouns for RMO

Identified noun	Notes on including noun as a thing to store
Accounting	We know who they are. No need to store it.
Back order	A special type of order? Or a value of order status? Research.
Back-order information	An output that can be produced from other information.
Bank	Only one of them. No need to store.
Catalog	Yes, need to recall them, for different seasons and years. Include.
Catalog activity reports	An output that can be produced from other information. Not stored.
Catalog details	Same as catalog? Or the same as product items in the catalog? Research.
Change request	An input resulting in remembering changes to an order.
Charge adjustment	An input resulting in a transaction.
Color	One piece of information about a product item.
Confirmation	An output produced from other information. Not stored.
Credit card information	Part of an order? Or part of customer information? Research.
Customer	Yes, a key thing with lots of details required. Include.
Customer account	Possibly required if an RMO payment plan is included. Research.
Fulfillment reports	An output produced from information about shipments. Not stored.
Inventory quantity	One piece of information about a product item. Research.
Management	We know who they are. No need to store.
Marketing	We know who they are. No need to store.
Merchandising	We know who they are. No need to store.

identifier or key an attribute the value of which uniquely identifies an individual thing or object

compound attribute an attribute that consists of multiple pieces of information but is best treated in the aggregate

association a term, in UML, that describes a naturally occurring relationship between specific things

attribute may be used to identify a specific thing, such as a Social Security number for an employee or an order number for a purchase. The attribute that uniquely identifies the thing is called an **identifier or key**. Sometimes, the identifier is already established (a Social Security number, vehicle ID number, or product ID number). Sometimes, the system needs to assign a specific identifier (an invoice number or transaction number).

A system may need to remember many similar attributes. For example, a customer has several names: a first name, a middle name, a last name, and possibly a nickname. A **compound attribute** is an attribute that contains a collection of related attributes, so an analyst may choose one compound attribute to represent all these names, perhaps naming it *Customer full name*. A customer might also have several phone numbers: home phone number, office phone number, fax phone number, and cell phone number. The analyst might start out by describing the most important attributes but later add to the list. Attribute lists can get quite long. Some examples of attributes of a customer and the values of attributes for specific customers are shown in **Figure 4-3**.

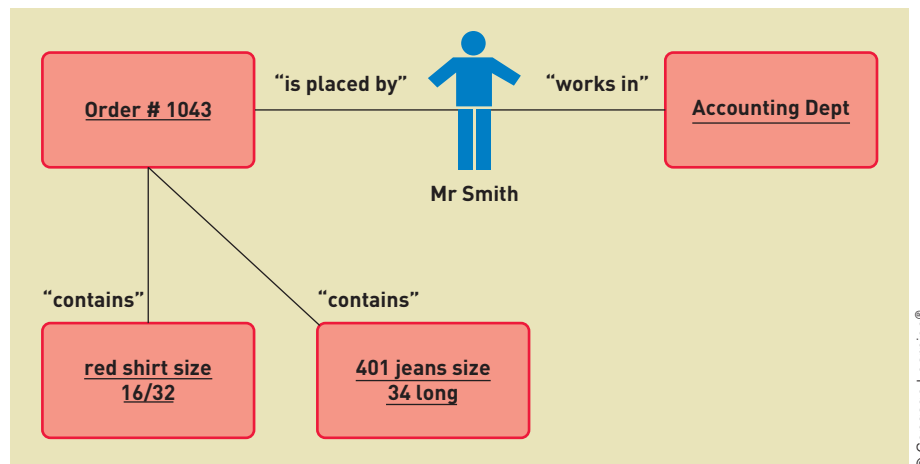
■ Associations among Things

After recording and refining the list of things and determining potential attributes, the analyst needs to research and record additional information. Many important relationships among things are important to the system. An **association** is a naturally occurring relationship between specific things, such as *an order is placed by a customer* and *an employee works in a department* (see **Figure 4-4**). *Is placed by* and *works in* are two associations that naturally occur between specific things. Information systems need to store information about things such as employees and departments, but equally important is storing information about the specific associations between those things; for

FIGURE 4-3 Attributes and values

All customers have these attributes:	Each customer has a value for each attribute:		
Customer ID	101	102	103
First name	John	Mary	Bill
Last name	Smith	Jones	Casper
Home phone	555-9182	423-1298	874-1297
Work phone	555-3425	423-3419	874-8546

FIGURE 4-4 Associations naturally occur among things



relationship a term, in database management, that describes a naturally occurring association between specific things

cardinality a measure of the number of links in a particular relationship between a thing (database data entity) and one or more other things (database data entities)

multiplicity in UML, a measure of the number of links in a particular association between a thing (object) and one or more other things (objects)

multiplicity constraints the actual numeric count of the constraints on things (UML objects) allowed in an association

example, John works in the Accounting Department and Mary works in the Marketing Department. Similarly, it is quite important to store the fact that Order 1043 for a shirt was placed by John Smith. In database management, the term **relationship** is often used in place of association, which is the term used when modeling in Unified Modeling Language (UML). This book uses the term *association* because it emphasizes UML diagrams and terms.

Associations between things apply in two directions. For example, *a customer places an order* describes the association in one direction. Similarly, *an order is placed by a customer* describes the association in the other direction. It is important to understand the association in both directions because sometimes it might seem more important for the system to record the association in one direction than in the other. For example, Ridgeline Mountain Outfitters definitely needs to know what items a customer ordered so the shipment can be prepared. However, it might not be initially apparent that the company needs to know about the customers who have ordered a particular item. What if the company needs to notify all customers who ordered a defective or recalled product? Knowing this information would be very important, but the operational users might not immediately recognize that issue.

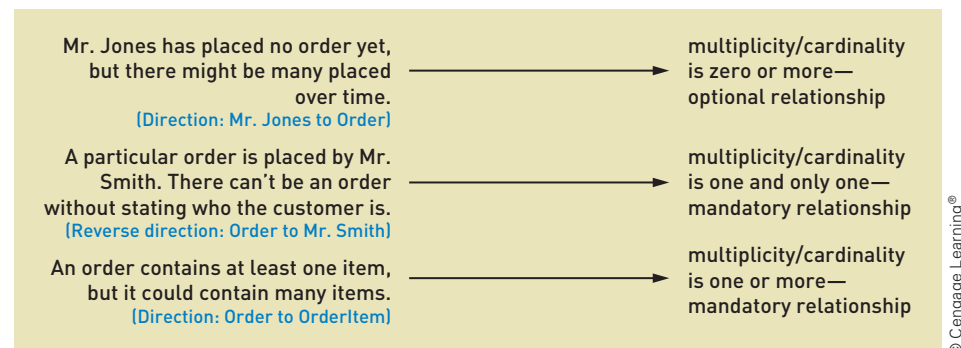
It is also important to understand the nature of each association in terms of the number of links for each thing and in which direction. For example, a customer might place many different orders, but an order is placed by only one customer. In database management, the number of links that occur is referred to as the **cardinality** of the association. Cardinality can be one-to-one or one-to-many. The term **multiplicity** is used to refer to the number of links in UML and should be used when discussing UML models. Multiplicity is established for each direction of the association. **Figure 4-5** lists examples of cardinality/multiplicity associated with an order.

It is important to describe not just the multiplicity but also the range of possible values of the multiplicity (the minimum and maximum multiplicity). For example, a particular customer might not ever place an order. In this case, there are zero associations. Alternatively, the customer might place one order, meaning one association exists. Finally, the customer might place two, three, or even more orders. The relationship for a customer placing an order can have a range of zero, one, or more, usually indicated as zero or more. The *zero* is the minimum multiplicity, and *more* is the maximum multiplicity. These terms are referred to as **multiplicity constraints**.

In some cases, at least one association is required (a mandatory as opposed to optional association). For example, the system might not record any information about a customer until the customer places an order. Therefore, the multiplicity would read *customer places one or more orders*.

A one-to-one association can also be refined to include minimum and maximum multiplicity. For example, an order is placed by one customer; it is impossible to have an order if there is no customer. Therefore, one is the minimum

FIGURE 4-5 Multiplicity/cardinality of associations



binary associations associations between exactly two distinct types of things

unary association an association between two instances of the same type of thing

ternary association an association between exactly three distinct types of things

***n*-ary association** an association between *n* distinct types of things

multiplicity, making the association mandatory. Because there cannot be more than one customer for each order, one is also the maximum multiplicity. Sometimes, such an association is read as *an order must be placed by one and only one customer*.

The associations described here are between two different types of things—for example, a customer and an order. These are called **binary associations**. Sometimes, an association is between two things of the same type—for example, the association *is married to*, which is between two people. This type of association is called a **unary association** (and sometimes called a recursive association). Another example of a unary association is an organizational hierarchy in which one organizational unit reports to another organizational unit—the Packing Department reports to Shipping, which reports to Distribution, which reports to Marketing.

An association can also be among three different types of things, when it is called a **ternary association**, or among any number of different types of things, when it is called an ***n*-ary association**. For example, one particular order might be associated with a specific customer plus a specific sales representative, requiring a ternary association.

■ The Entity-Relationship Diagram

data entities the term used in ERD modeling to describe things about which the system needs to store information

entity-relationship diagram (ERD) a diagram consisting of data entities, their attributes, and their relationships

Traditional approaches to system development place a great deal of emphasis on data requirements for a new system and use the term **data entities** for the things about which the system needs to store information. Data requirements include the data entities, their attributes, and the relationships (called *associations* in UML) among the data entities. The brainstorming technique and noun technique described earlier are used in the same way with data entities as with domain classes used for UML modeling. A model commonly used by traditional analysts and database analysts is called the **entity-relationship diagram (ERD)**. The ERD is not a UML diagram, but it is frequently used and is quite similar to the UML domain model class diagram that is discussed later in this chapter.

■ Examples of ERD Notation

On the entity-relationship diagram, rectangles represent data entities, and the lines connecting the rectangles show the relationships among data entities. **Figure 4-6** shows an example of a simplified entity-relationship diagram with two data entities: Customer and Order. Each Customer can place many Orders, and each Order is placed by one Customer. The cardinality is one-to-many in

FIGURE 4-6 A simple entity-relationship diagram (ERD)

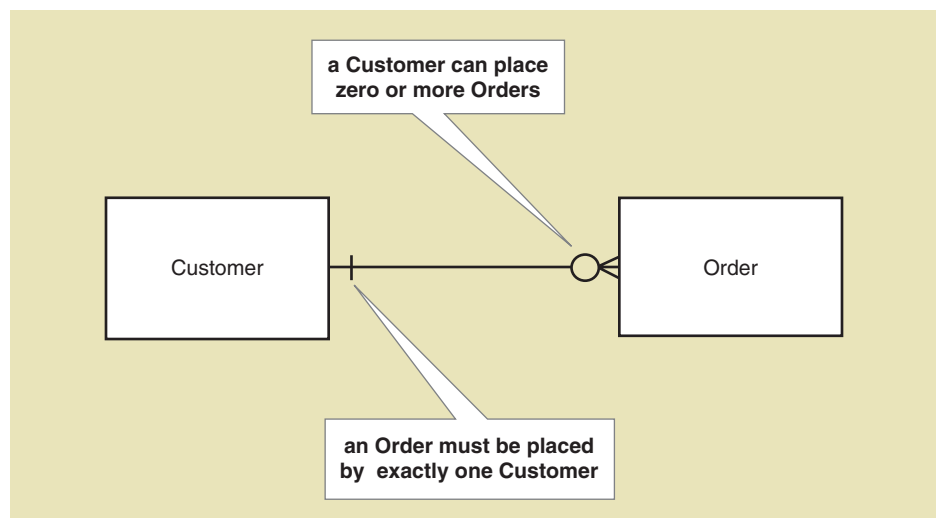
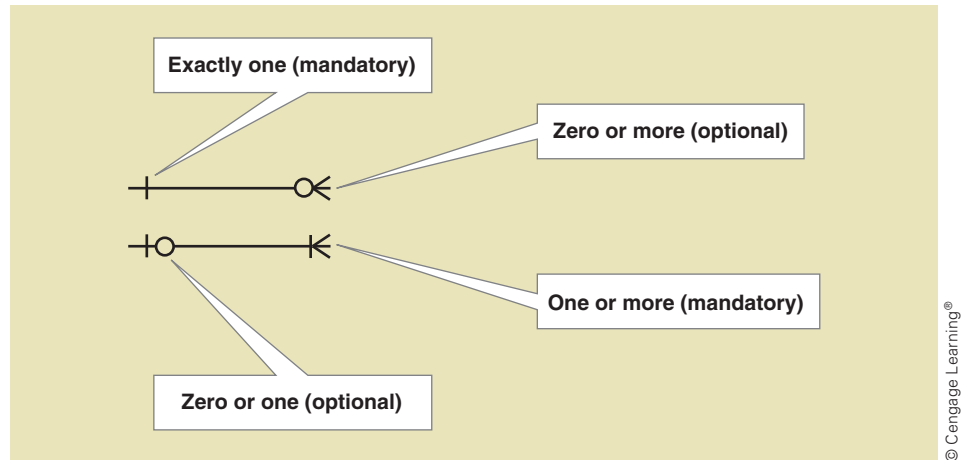


FIGURE 4-7 Cardinality symbols of ERD relationships



one direction and one-to-one in the other direction. The crow's-foot symbol on the line next to the Order data entity indicates many orders. But other symbols on the relationship line also represent the minimum and maximum cardinality constraints. See Figure 4-7 for an explanation of ERD relationship symbols. The model in Figure 4-6 actually says that a Customer places a minimum of zero Orders and a maximum of many Orders. Reading in the other direction, the model says an Order is placed by at least one and only one Customer.

Cardinality constraints reflect the business policies defined by management. The analyst does not decide that two customers cannot share one order. Rather, the analyst must discover the underlying policy and accurately represent it using cardinality constraints. The analyst must be very careful not to invent policies by defining arbitrary cardinality constraints. When the discovery process raises policy questions that haven't yet been answered—the analyst must raise the question with management and let it decide on an appropriate policy.

Figure 4-8 shows the model expanded to include the order items (one or more specific items included on the order). Each order contains a minimum of one and a maximum of many items (there could not be an order if it did not contain at least one item). For example, an order might include a shirt, a pair of shoes, and a belt, and each of these items is associated with the order. This example also shows some of the attributes of each data entity: A customer has a customer number, a name, a billing address, and several phone numbers. Each order has an order ID, order date, and so on. Each order item has an item ID, quantity, and price. The attributes of the data entity are listed below the name, with the key identifier listed first, usually followed by *PK* to indicate primary key.

Figure 4-9 shows how the actual data in some transactions might look. John is a customer who has placed two orders. The first order, placed on February 4, was for two shirts and one belt. The second order, placed on March 29, was for one pair of boots and two pairs of sandals. Mary is a customer who has not yet placed an order. Recall that a customer might place zero or more orders. Therefore, Mary is not associated with any orders. Finally, Sara placed an order on March 30 for three pairs of sandals. The diagram shown in Figure 4-9 is

FIGURE 4-8 An expanded ERD with attributes shown

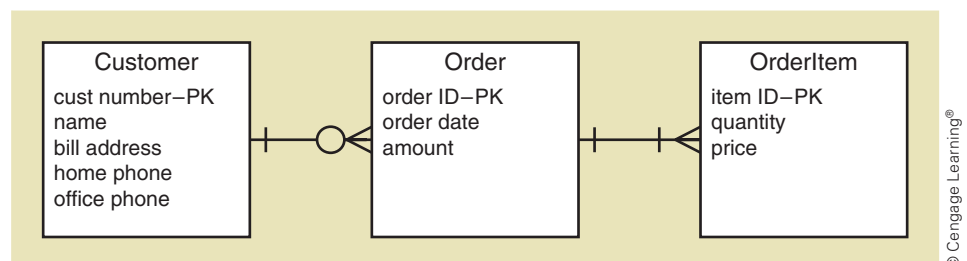
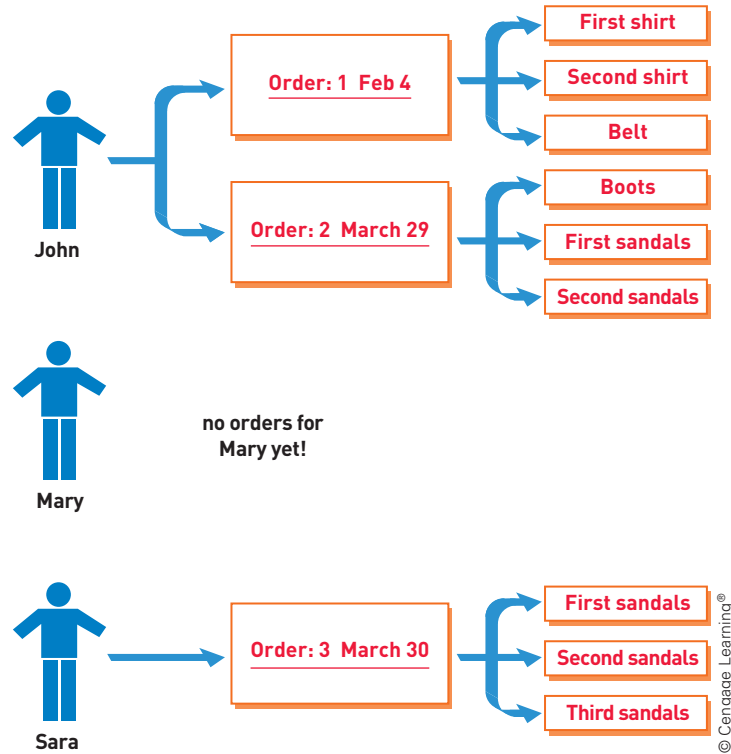


FIGURE 4-9 Semantic net of customers, orders, and order items consistent with the expanded ERD

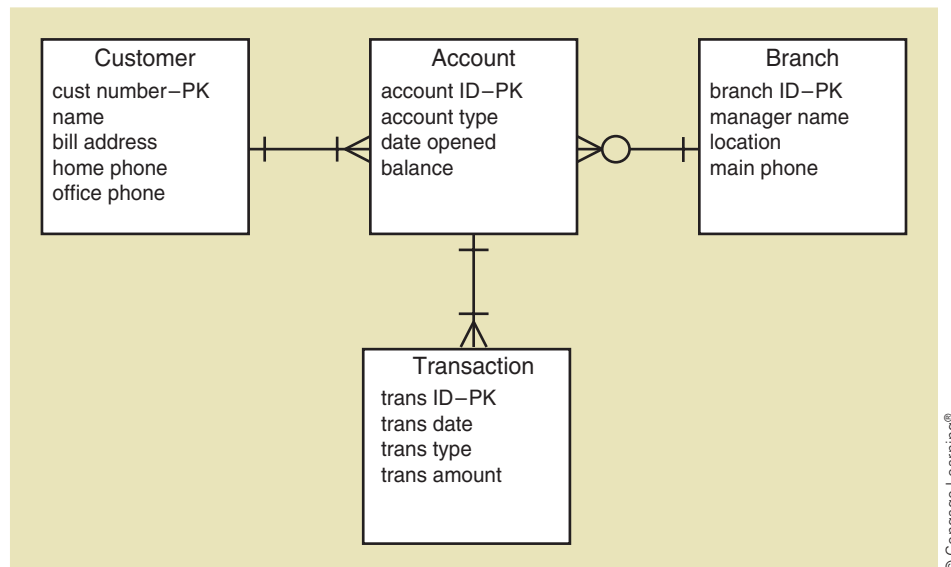


semantic net a graphical representation of an individual data entity and its relationship with other individual data entities

sometimes referred to as a semantic net. A **semantic net** shows specific objects that belong to a class or data entity and the links among them. A semantic net is useful for thinking through and verifying the entities and relationships in an ERD and the classes and associations in a class diagram (discussed next).

Another example is shown in **Figure 4-10**. This ERD is for a bank that has many branches. Each branch has one or more accounts. Each account is owned by one customer and results in one or more transactions. There are a few other issues to consider in the bank example. First, there is no data entity named Bank. That is because the ERD shows data storage requirements for the bank. There is only one bank. Therefore, there is no need to include Bank in the model. This is a general rule that applies to ERDs. If the system were for state bank regulators, then Bank would be an important data entity because there are lots of banks under the state regulators' jurisdiction.

FIGURE 4-10 An ERD for a bank with many branches



Look again at the cardinality. Note that a customer must have at least one account. The rationale for this is that the bank would not add a customer unless he or she was adding an account. Note also that the branch can have zero accounts. A branch might be added long before it opens its doors, so it is possible that it does not have any accounts. Additionally, there might be some branches that do not have accounts, such as a kiosk at a university or airport. Note that an account must have at least one transaction. The rationale is that opening a new account requires an initial deposit, which is a transaction. It is important to recognize that questions about the cardinality and minimum and maximum cardinality constraints need to be discussed and reviewed with stakeholders.

■ The Domain Model Class Diagram

class a category or classification of a set of objects or things

domain classes classes that describes objects from the problem domain

class diagram a diagram consisting of classes (i.e., sets of objects) and associations among the classes

domain model class diagram a class diagram that only includes classes from the problem domain

camelback or camelcase notation when words are concatenated to form a single word and the first letter of each embedded word is capitalized

As discussed previously, many current approaches to system development use the term *domain class* rather than *data entity* and use concepts and notations based on UML to model the things in the problem domain. These concepts come from the object-oriented approach to system development. A **class** is a category or classification used to describe a collection of objects. Each object belongs to a class. Therefore, students Mary, Joe, and Maria belong to the class Student. Classes that describe things in the problem domain are called **domain classes**. Domain classes have attributes and associations. Multiplicity (called cardinality in an ERD) applies among classes. Initially, when defining requirements, the approach to modeling is very similar whether the analyst is using ERD or UML diagrams.

The UML **class diagram** is used to show classes of objects for a system. One type of UML class diagram that shows the things in the users' problem domain is called the **domain model class diagram**. Another type of UML class diagram is called the design class diagram, and it is used when designing software classes. You will learn about the design class diagram in Chapter 12.

On a class diagram, rectangles represent classes, and the lines connecting the rectangles show the associations among classes. **Figure 4-11** shows such a symbol for a single domain class: Customer. The domain class symbol is a rectangle with two sections. The top section contains the name of the class, and the bottom section lists the attributes of the class. Later, you will learn that the design class symbol includes a third section at the bottom for listing methods of the class; methods do not apply to problem domain classes.

Class names and attribute names use **camelback** (sometimes called **camelcase**) notation, in which the words run together without a space or underscore. Class names begin with a capital letter; attribute names begin with a lowercase letter (see Figure 4-11). Class diagrams are drawn by showing classes and associations among classes. The examples used previously for the entity-relationship diagram are redrawn by using UML domain class diagram notation in the following section so you can compare them. Additionally, more complex issues about classes and associations can be illustrated by using domain model class diagrams.

FIGURE 4-11 The UML domain class symbol with name and attributes

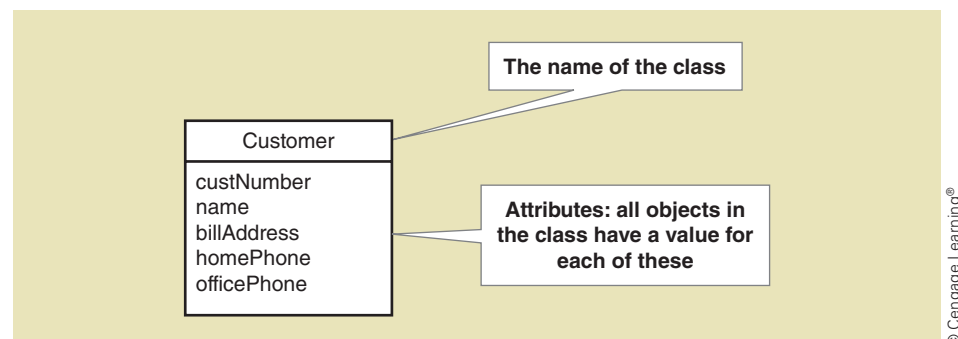
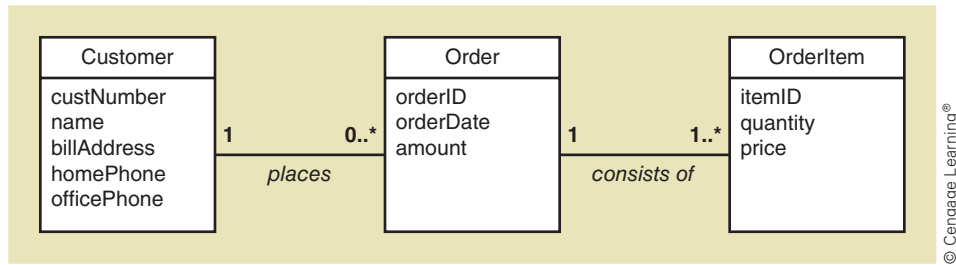


FIGURE 4-12 A simple domain model class diagram



■ Domain Model Class Diagram Notation

Figure 4-12 shows a simplified domain model class diagram with three classes: Customer, Order, and OrderItem (just like the example of an ERD shown in Figure 4-8). Here, each class symbol includes two sections. In diagram notation, you see that each Customer can place many Orders (a minimum of zero and a maximum of many) and that each Order is placed by one Customer. The associations *places* and *consists of* can be included on the diagram for clarity, as shown in Figure 4-12, but this detail is optional. The multiplicity is one-to-many in one direction and one-to-one in the other direction. The multiplicity notation, shown as an asterisk on the line next to the Order class, indicates many orders. The other association shows that an Order consists of one or more OrderItems, and each OrderItem is associated with one Order.

See Figure 4-13 for a summary of multiplicity notation.

Figure 4-14 shows another example of a domain model class diagram, this one for the bank with multiple branches that was discussed earlier and shown as an ERD. In this example, the UML notation for indicating an attribute that is an identifier or key is {key}.

Figure 4-15 shows an example of a domain model class diagram with a many-to-many association. At a university, courses are offered as course sections, and a student enrolls in many course sections. Each course section contains many students. Therefore, the association between CourseSection and Student is many-to-many. There are situations in which many-to-many associations are appropriate, and they can be modeled as shown.

On closer analysis, analysts often discover that many-to-many associations involve additional data that are important and must be stored. For example, in Figure 4-15, where is the grade that each student receives for the course stored? This is important data, and although the model indicates which course section a student took, it does not have a place for the grade. Grade isn't an attribute of CourseSection alone. Nor is it an attribute of Student alone. Rather, it's an attribute of the association between CourseSection and StudentGrade.

FIGURE 4-13 UML notation for multiplicity of associations

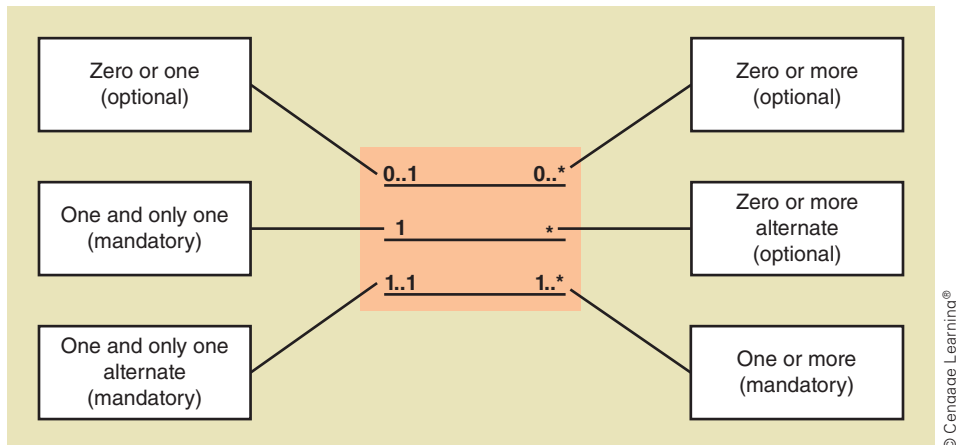


FIGURE 4-14 A domain model class diagram for a bank

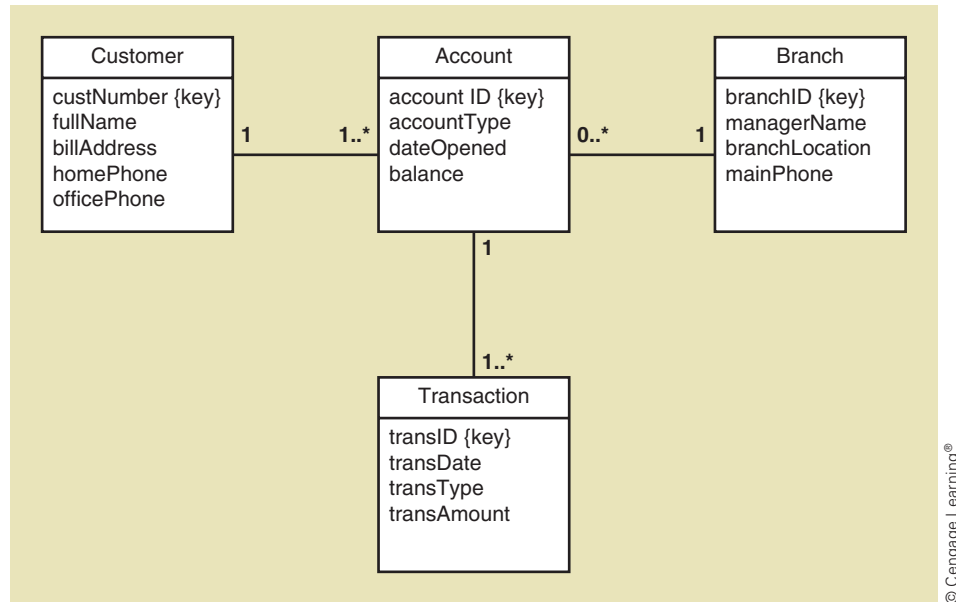
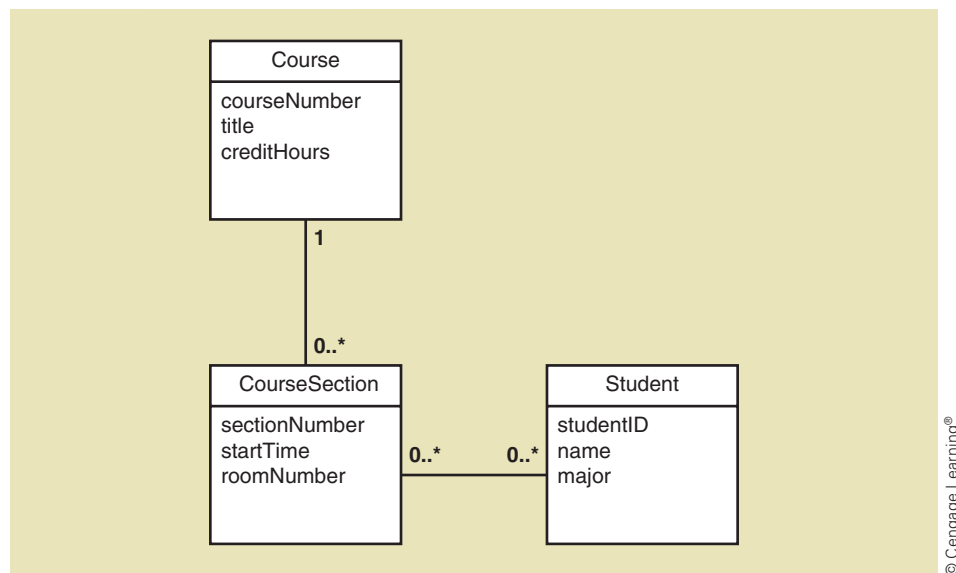


FIGURE 4-15 A university course enrollment domain model class diagram with a many-to-many association



association class an association that is also treated as a class; often required in order to capture attributes for the association

Adding a domain class, called an **association class**, to represent the association between Student and CourseSection provides a place in which to store the missing attribute for grade. Figure 4-16 shows the expanded class diagram, with an association class named CourseEnrollment, which has an attribute for the student's grade. A dashed line connects the association class with the association line between the CourseSection and Student classes.

Reading the association in Figure 4-16 from left to right, one course section has many course enrollments—each with its own grade—and each course enrollment applies to one specific student. Reading from right to left, one student has many course enrollments—each with its own grade—and each course enrollment applies to one specific course section. A database implemented by using this model will be able to produce grade lists showing all students and their grades in each course section as well as grade transcripts showing all grades earned by each student.

FIGURE 4-16 A refined university course enrollment domain model class diagram with an association class

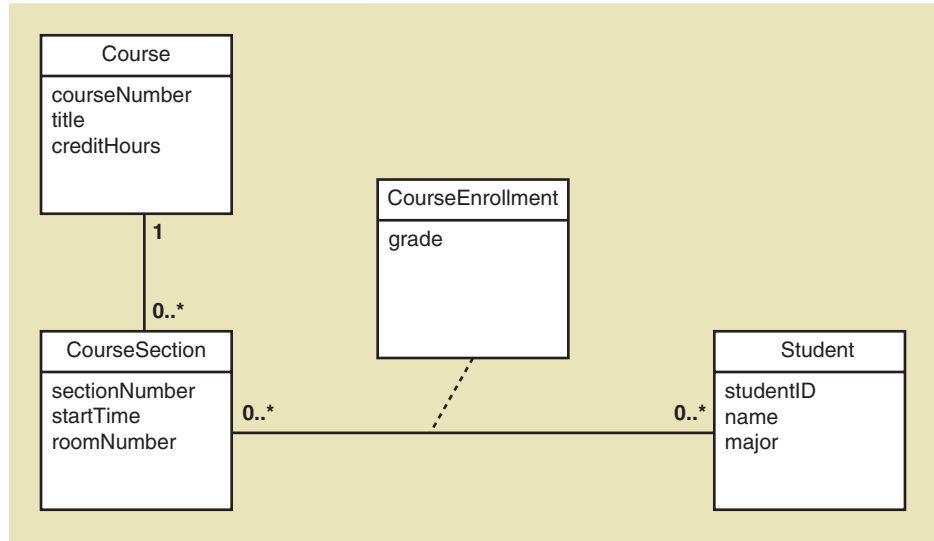
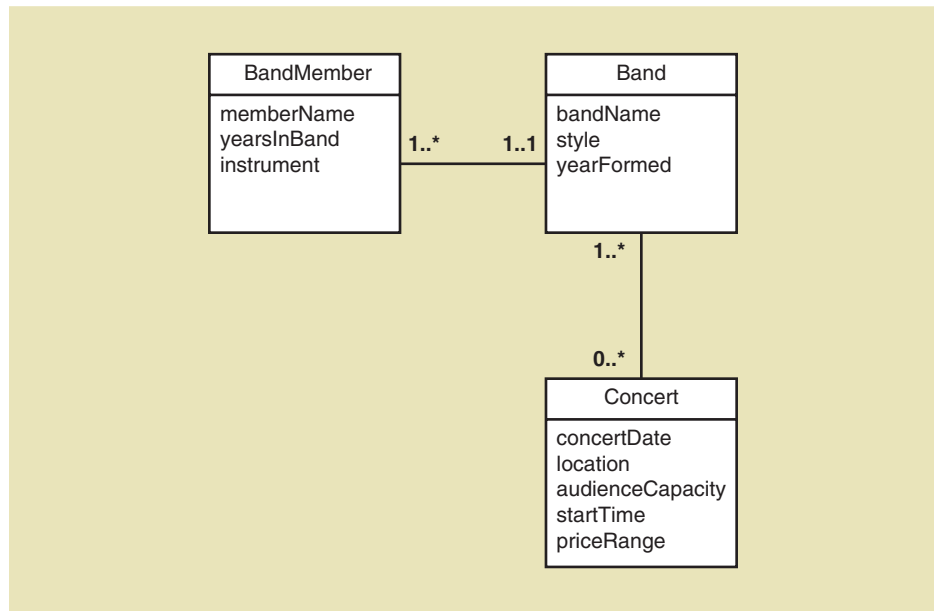


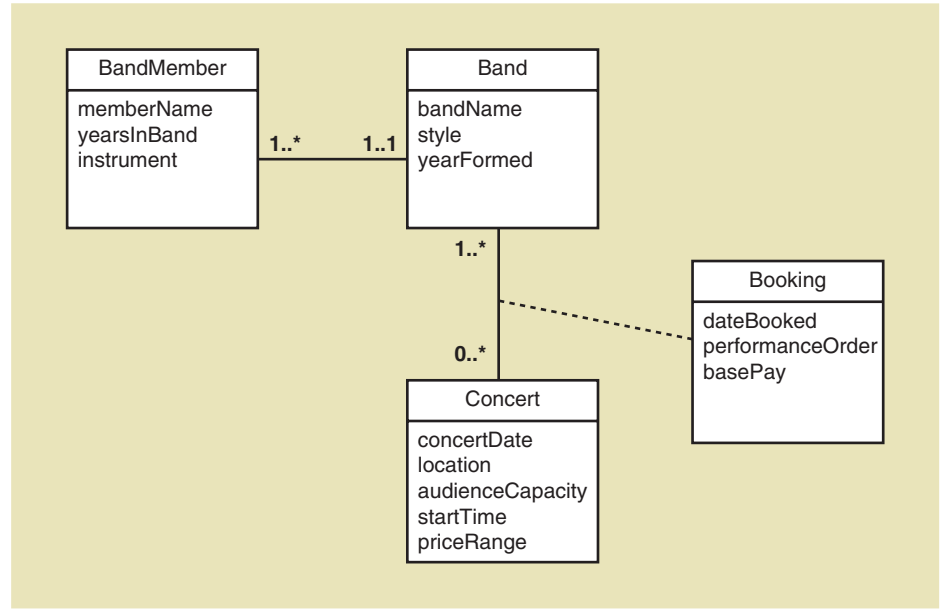
FIGURE 4-17 An initial band booking domain model class diagram with a many-to-many association



Another example of a domain model class diagram with a many-to-many association is shown in **Figure 4-17**. A band has one or more band members, and in this case a band member is in one and only one band. The many-to-many association is between the classes Band and Concert. A band is booked for zero or more concerts, and a concert books one or more bands. Note that on the minimum multiplicities, a band might not have any concerts booked at a given time, but a concert is ordinarily created only when at least one band is booked. These minimum multiplicities can seem somewhat arbitrary, although they do convey important information for programmers and database analysts. Be sure it is the policymakers who make the decisions about multiplicity constraints.

Looking more closely at the many-to-many association, the analyst might ask if there is some additional information about the booking that needs to be captured and remembered by the system. Is it important to store the base pay expected for the booking? Pay differs for each concert. Is it important to store the order of performances at the concert? Some booking might be main act and some might be opening act. **Figure 4-18** shows a refined domain model class diagram

FIGURE 4-18 A refined band booking domain model class diagram with an association class



that adds an associative class named Booking that includes attributes dateBooked, performanceOrder, and basePay. Reading from Band to Concert, the diagram says one band is booked July 7 as a main act for \$4,500 (from Booking class) for a concert on July 28 at the Downtown Concert House (from Concert class). Again reading from Band to Concert, the diagram says another band is booked July 9 as the opening act for \$2,500 (from Booking class) for a concert on July 28 at the Downtown Concert House (from Concert class).

■ More Complex Issues about Classes of Objects

Previously, you learned about associations between domain classes. In UML, an association is one of many types of relationships, so you need to be more precise when discussing UML diagrams than when discussing ERDs. With class diagrams, there are actually three types of relationships among classes of objects: association relationships (discussed previously), generalization/specialization relationships, and whole-part relationships. This section discusses generalization/specialization relationships and whole-part relationships and shows how they are represented in UML class diagrams.

■ Generalization/Specialization Relationships

generalization/specialization relationships

a type of hierarchical relationship in which subordinate classes are subsets of objects of the superior classes; an inheritance hierarchy

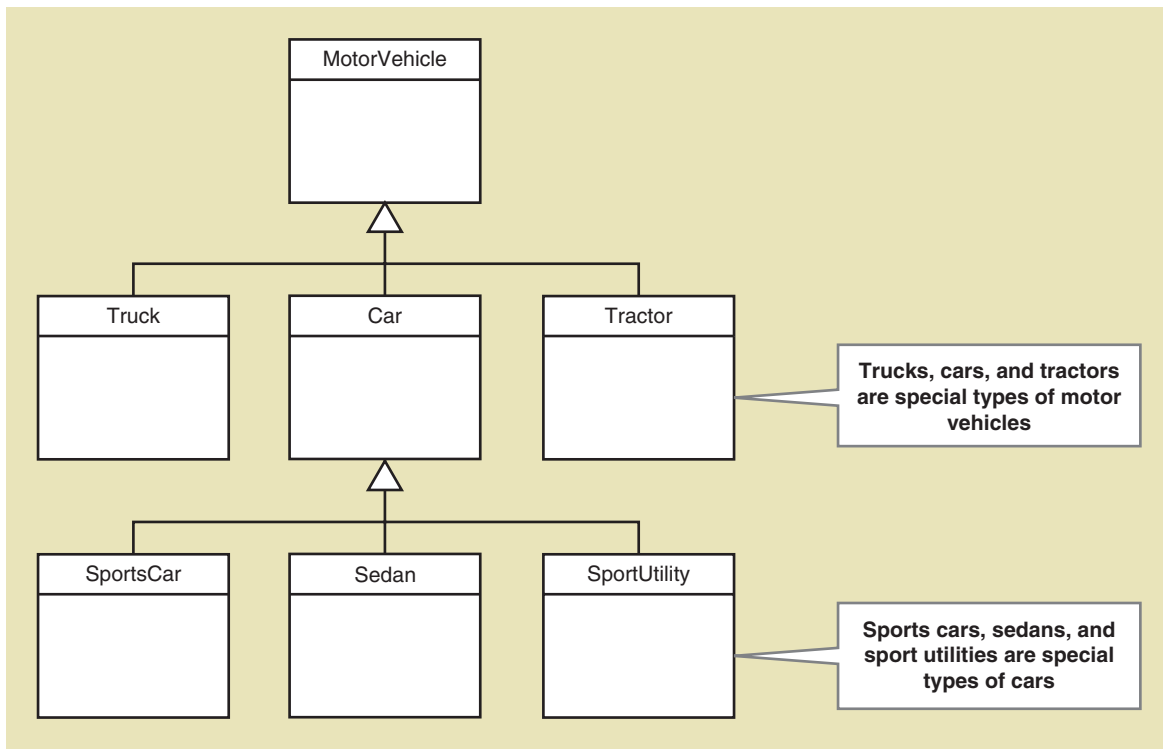
Generalization/specialization relationships are based on the idea that people classify things in terms of similarities and differences. Generalizations are judgments that group similar types of things. For example, there are several types of motor vehicles: cars, trucks, and tractors. All motor vehicles share certain general characteristics, so a motor vehicle is a more general class. Specializations are judgments that group different types of things. For example, special types of cars include sports cars, sedans, and sport utility vehicles. These types of cars are similar in some ways yet different in other ways. Therefore, a sports car is a special type of car.

A generalization/specialization relationship is used to structure or rank these things from the more general to the more special. As discussed previously, classification refers to defining classes of things. Each class of things in the hierarchy might have a more general class above it, called a **superclass**. At the same time, a class might have a more specialized class below it, called a **subclass**. In **Figure 4-19**, the class Car has three subclasses and one superclass (MotorVehicle). UML class diagram notation uses a triangle that points to the superclass to show a generalization/specialization hierarchy.

superclass the superior or more general class in a generalization/specialization relationship

subclass the subordinate or more specialized class in a generalization/specialization relationship

FIGURE 4-19 Generalization/specialization relationships for motor vehicles



© Cengage Learning®

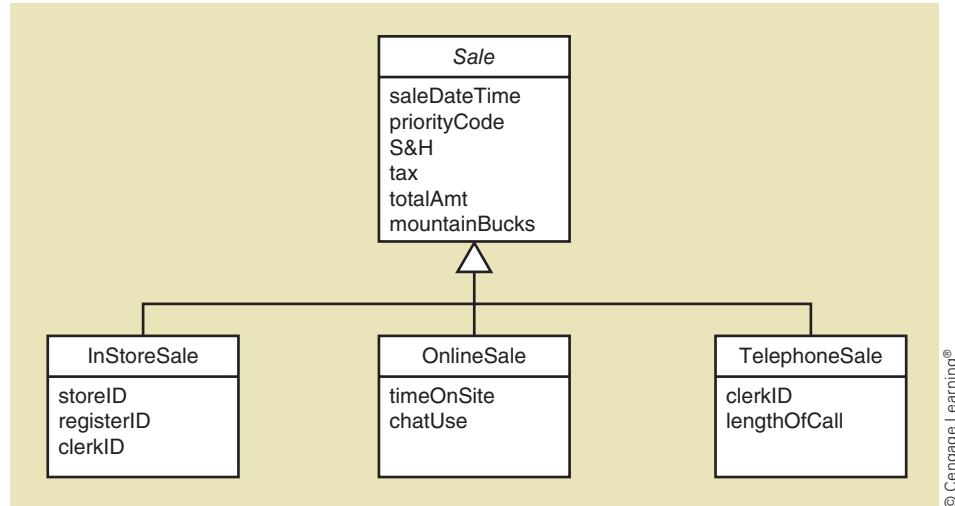
We mentioned that people structure their understanding by using generalization/specialization relationships. In other words, people learn by refining the classifications they make about some field of knowledge. A knowledgeable banker can talk at length about special types of loans and deposit accounts. A knowledgeable merchandiser like John Blankens at Ridgeline Mountain Outfitters can talk at length about special types of outdoor activities and clothes. Therefore, when asking users about their work, the analyst is trying to understand the knowledge the user has about the work, which the analyst can represent by constructing generalization/specialization relationships. At some level, the motivation for the new CSMS project at RMO started with John's recognition that Ridgeline Mountain Outfitters might handle many special types of sales with a new system (online sales, telephone sales, and in-store sales). These special types of sales are shown in **Figure 4-20**.

inheritance the concept that specialization classes inherit characteristics of the generalization class

Inheritance allows subclasses to share characteristics of their superclass. Returning to Figure 4-19, a car is everything any other motor vehicle is but also something special. A sports car is everything any other car is but also something special. In this way, the subclass "inherits" the characteristics of the superclass. In the object-oriented approach, inheritance is a key concept that is possible because of generalization/specialization hierarchies. Sometimes, these are referred to as inheritance relationships. Another term often used to describe generalization/specialization is an "ISA" relationship. In other words, it makes sense to say, "a Sedan IS A special type of Car and a Car IS A special type of MotorVehicle."

Consider Figure 4-20, which represents a partial domain model for a retail business such as RMO. Attributes are included for each class in the hierarchy. Each member of the Sale class has a `saleDateTime` attribute and a `priorityCode` attribute. Each `InStoreSale` has a `storeID`, `clerkID`, and `registerID`, but an `OnlineSale` and a `TelephoneSale` have other attributes. `OnlineSale`, `InStoreSale`, and `TelephoneSale` inherit the attributes from Sale, plus they have some special attributes of their own. An `OnlineSale` actually has eight attributes (six from Sale and two additional). An `InStoreSale` has nine attributes, and a `TelephoneSale` has eight attributes.

FIGURE 4-20 Generalization/specialization relationships (inheritance) for sales



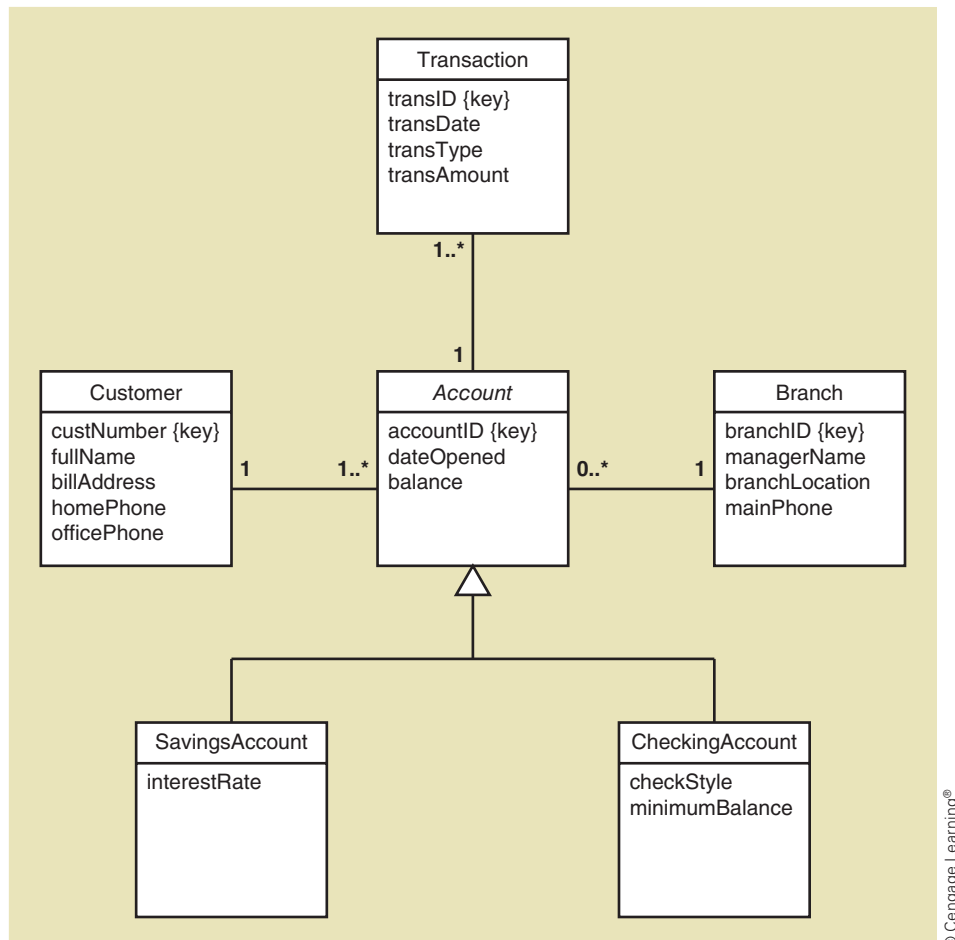
abstract class a class that only exists in a model so subclasses can inherit from it

concrete class a class that allows individual objects or instances to exist

Note that in Figure 4-20 that the class name *Sale* is in italic; that is because it is an abstract class. An **abstract class** is a class that only exists so subclasses can inherit from it. There is never an actual object simply called a *Sale*. Each sale must be one of the three subclasses. A **concrete class** is a class that does have actual objects. Sometimes, a superclass is abstract; sometimes, it is concrete depending on the intention of the analyst.

Figure 4-21 shows an extension of the bank with multiple branches example to indicate that there are two types of accounts: a *SavingsAccount* and a *CheckingAccount*. The abstract class *Account* is in italic, indicating it is an abstract

FIGURE 4-21 An expanded domain model class diagram for the bank, with subclasses for types of accounts



class. Rather than including an attribute for account type, the subclasses represent different types of accounts. Each subclass has its own special attributes that do not apply to the other subclasses. A SavingsAccount has four attributes, and a CheckingAccount has five attributes. Note that each subclass also inherits an association with a Customer, optionally a Branch, and one or more Transactions.

■ Whole-Part Relationships

Another way that people structure information about things is by defining them in terms of their parts. For example, learning about a computer system might involve recognizing that the computer is actually a collection of parts: processor, main memory, keyboard, disk storage, and monitor. A keyboard is not a special type of computer; it is part of a computer, but it is also something separate.

Whole-part relationships are used to show an association between one class and other classes that are parts of that class.

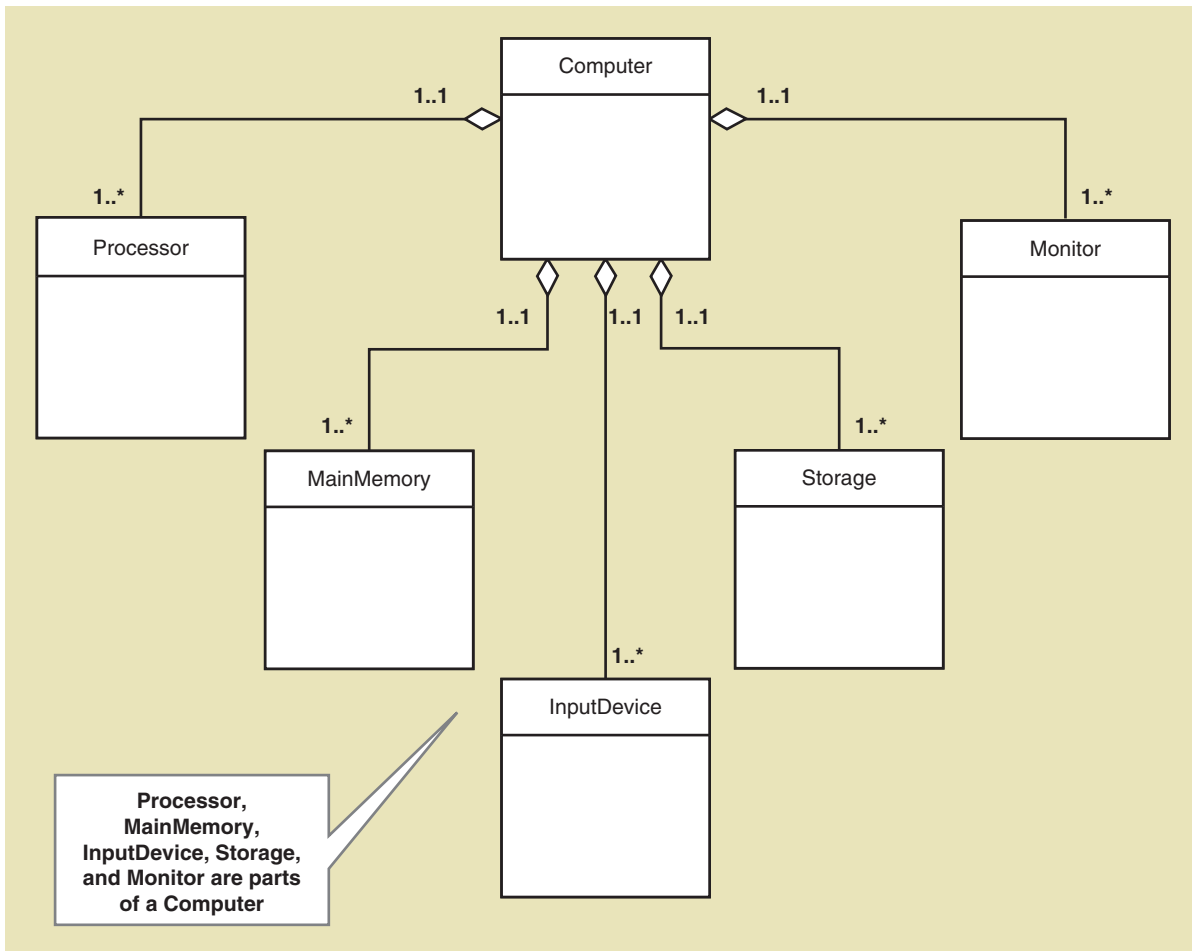
There are two types of whole-part relationships: aggregation and composition. **Aggregation** refers to a type of whole-part relationship between the aggregate (whole) and its components (parts), where the parts can exist separately. **Figure 4-22** demonstrates the concept of aggregation in a computer system, with the UML diamond symbol representing aggregation. **Composition** refers to whole-part relationships that are even stronger, where the parts, once associated, can no longer exist separately. The UML diamond symbol is filled in solid to represent composition. For example, think of a house that is made up of bathrooms, bedrooms, stairwell, and so forth. The bathrooms or bedrooms

whole-part relationships relationships between classes in which one class is a part or a component portion of another class

aggregation a type of whole-part relationship in which the component parts also exist as individual objects apart from the aggregate

composition a type of whole-part relationship in which the component parts cannot exist as individual objects apart from the total composition

FIGURE 4-22 Whole-part (aggregation) relationships between a computer and its parts



© Cengage Learning®

never exist apart from the house; therefore, this whole-part relationship is a composition and uses solid diamond connectors.

Whole-part relationships—aggregation and composition—mainly allow the analyst to express subtle distinctions about associations among classes. As with any association relationship, multiplicity can apply, such as when a computer has one or more storage devices.

The UML class diagram examples you have seen so far are domain model class diagrams. The design class diagram is a refinement of the class diagram and is used to represent software classes in the new system. You will learn about the process of converting the domain model class diagram to a design class diagram in Chapter 12.

■ The Ridgeline Mountain Outfitters Domain Model Class Diagram

The RMO CSMS involves many domain classes and many complex association and generalization/specialization relationships. A domain model class diagram for an information system evolves as the project proceeds; and unlike the use case diagrams, where many diagrams are created, there is eventually only one domain model class diagram. Also, unlike the use case diagram, the domain model class diagram is not produced just for presentations. The process of developing and refining the domain model class diagram is how the analyst explores and learns about the problem domain. Therefore, the information depicted in the domain model class diagram is very detailed and rich in specific meaning.

The Ridgeline Mountain Outfitters domain model class diagram is a variation of the customer, order, and order item example shown in Figure 4-12. Most of the domain classes are from the list of nouns developed in Figure 4-2. Because the model is fairly complex, an analyst might start by focusing on one subsystem at a time to reduce the complexity. Eventually, all subsystems can be combined into one domain model.

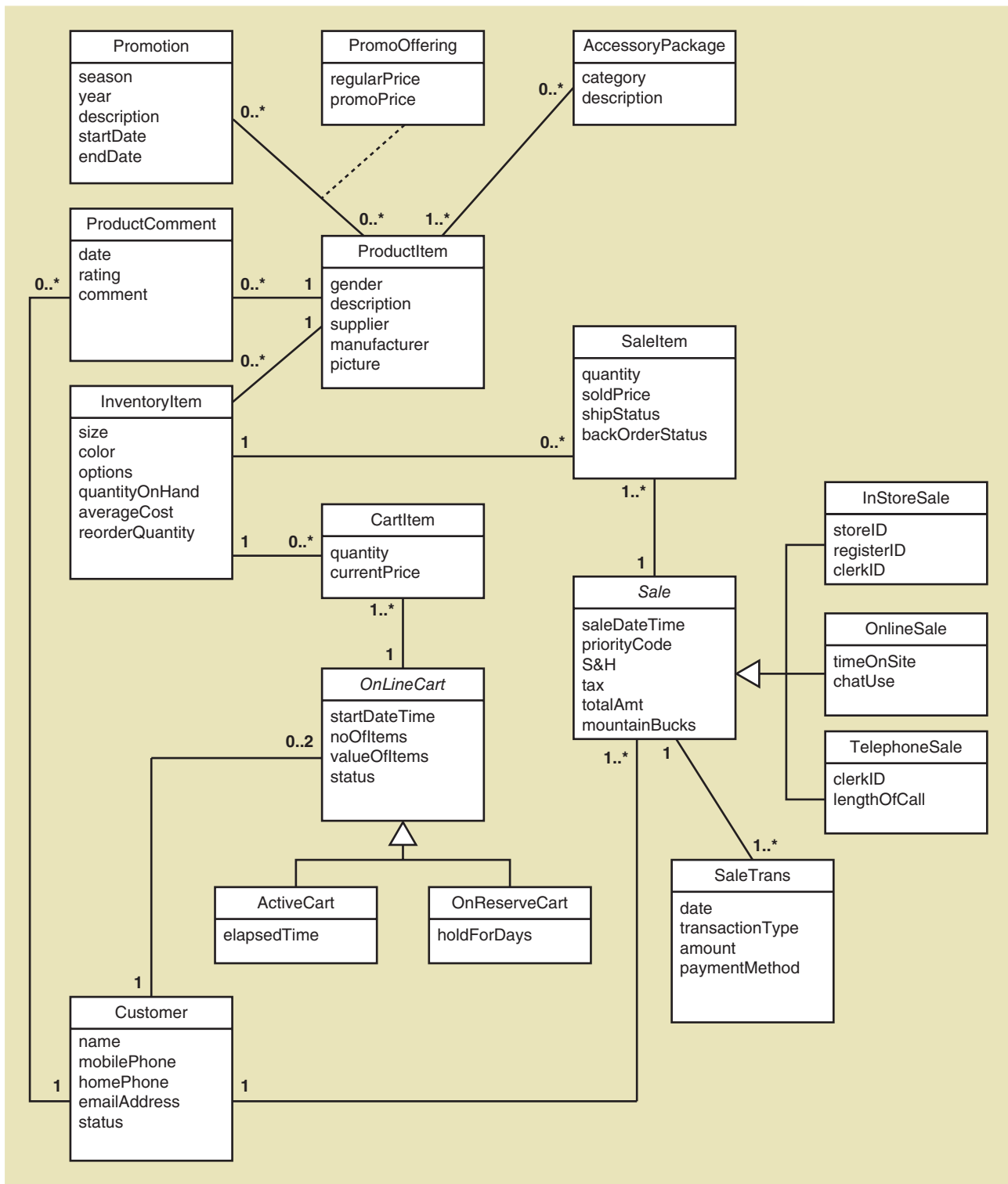
■ The RMO Sales Subsystem

Figure 4-23 shows a domain model class diagram for the RMO CSMS Sales subsystem. The Sales subsystem mainly involves the customer, sale, sales items, products, promotions, and accessories. That is a good starting point, but there are also additional domain classes. Additionally, recall that the association relationships are just as important as the classes, so these must be identified. There are also special types of sales and a shopping cart.

In Figure 4-23, each customer can be associated with one or more sales. Note that there are three special types of sales shown in the inheritance relationships (in-store sale, online sale, and telephone sale), as discussed in the last section. Therefore, the scope of the Sales subsystem includes in-store, online, and telephone sales processes. A customer can also be associated with an online shopping cart (OnLineCart) for any online sale. There are two special types of carts: the active cart and the on-reserve cart. The minimum multiplicity between customer and cart is zero, meaning there might not be a shopping cart involved—for example, in an in-store or telephone sale. There can be a maximum of two carts for a customer at any one time: an active cart and an on-reserve cart. The on-reserve cart can be remembered from session to session. Each sale and each cart is associated with one customer, so the subclasses inherit the association, just as they inherit the attributes of Sale.

Note that an individual sale is associated with one or more sale items. In the online cart, it is associated with one or more cart items. With an online sale, the sale is created from the cart when the customer checks out. Sale items are created from each cart item. Finally, a sales transaction is created and associated with the sale.

FIGURE 4-23 RMO Sales subsystem domain model class diagram



A sale can have one or more sale items, but what is each item? An association between each sale item and an inventory item answers the question. Each sale item is for a specific inventory item, meaning a specific size and color of the item, such as a shirt or coat. An inventory item has an attribute for the quantity on hand of that size and color. Because there are many colors and sizes (each with its own quantity), each inventory item is associated with a product item

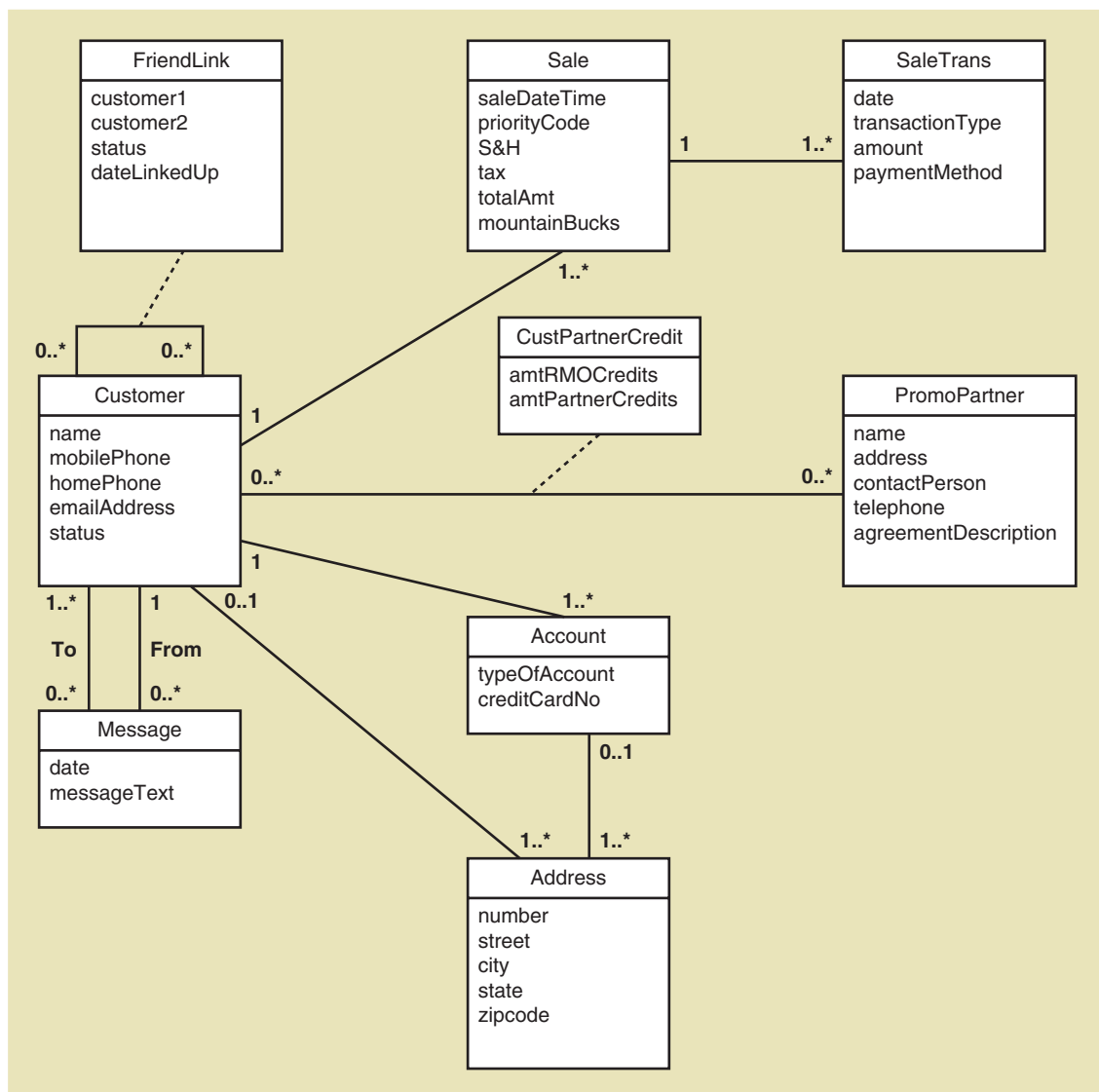
that describes the item generally (gender, description, supplier, manufacturer, and picture). Each product item is associated with many inventory items, and each inventory item is associated with many sale items.

A product item can be part of many promotions, and a promotion can include many product items, making a many-to-many association. An association class is added to store information about the price of each item in each promotion. Each product item might have many accessories, and an accessory might apply to many product items. Here, there is no defined association class for the many-to-many association. Note that this association might also be modeled as a unary (recursive) association. Finally, each product item can have many customer comments, which are reviewed during a sale.

■ The RMO Customer Account Subsystem

The Customer Account subsystem domain model class diagram is shown in **Figure 4-24**. Note that there are some classes repeated here that are also on the Sales subsystem domain model class diagram. For example, Customer is important to both subsystems. Note that Sale and SaleTrans are also included here. To make account adjustments and report on all payments and returns

FIGURE 4-24 RMO Customer Account subsystem domain in model class diagram



for a customer, all sales and sales transactions need to be referenced. Repeating domain classes in several subsystems does not mean there is redundancy. In complex domain models, it is easier to do the modeling and analysis in separate diagrams before merging them all together. Sometimes, the project team divides the work by subsystem, so each would work on a separate diagram being sure to coordinate with each other.

The Customer Account subsystem includes messages, partner credits, and friend links. The FriendLink class is an association class, but unlike other examples, it is attached to a unary association between customers. Each customer can be linked to many other customers, shown by the association line at the top of the Customer class. For each link, the status and dateLinkedUp is stored. The Message class is handled differently. Each customer can send many messages, each to many other customers. Similarly, each customer can receive many messages.

■ The Complete RMO Domain Model Class Diagram

The analysts at RMO may continue to model each subsystem separately. The exercises at the end of this chapter ask you to create those other subsystem diagrams. The final domain model class diagram for the RMO CSMS is shown in **Figure 4-25**. Classes not shown before include Shipper, Shipment, ReturnItem, and Suggestion. You should spend some time to understand all of the classes and associations in this model. Real-world domain model class diagrams are often much more complex than this example.

■ The State Machine Diagram—Identifying Object Behavior

Sometimes, it is important for a computer system to maintain information about the status of problem domain objects. For example, a customer might want to know whether a particular sale has been shipped or a manager might want to know if a customer sale has been paid for. Thus, the system needs to be able to track the status of customer sales. When defining requirements, analysts need to identify and document which domain objects require status checking and which business rules determine valid status conditions. Referring back to RMO, an example of a business rule is that a customer sale shouldn't be shipped until it has been paid for.

The status condition for a real-world object is often referred to as the *state* of the object. Defined precisely, a **state** of an object is a condition that occurs during its life when it satisfies some criterion, performs some action, or waits for an event. For real-world objects, we mean the same thing whether we refer to the status condition of an object or talk about its state.

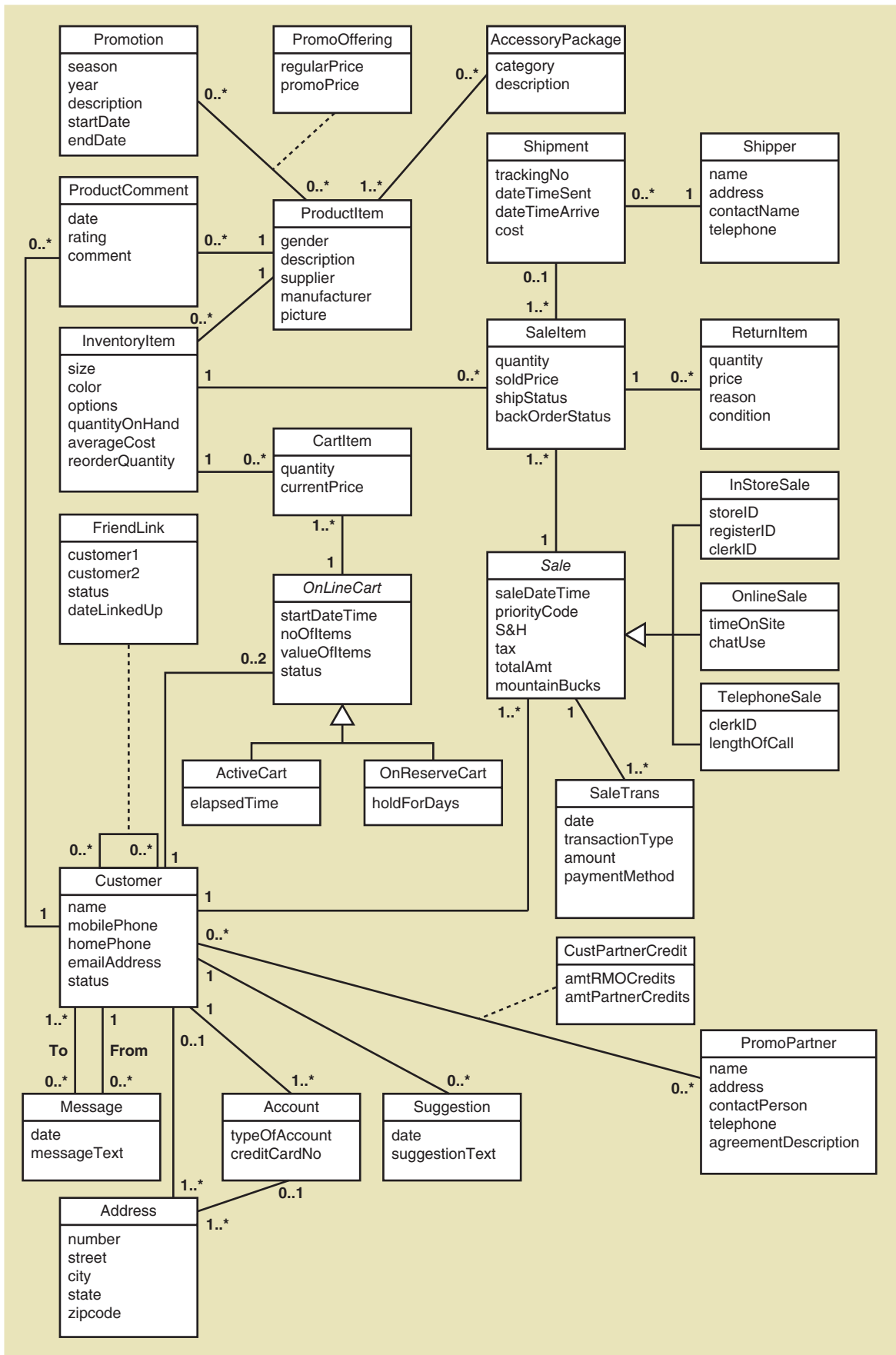
The naming convention for status conditions helps identify valid states. A state might have a name of a simple condition, such as *On* or *In repair*. Other states are more active, with names consisting of gerunds or verb phrases, such as *Being shipped* or *Working*. For example, a specific Sale object comes into existence when a customer buys something. Right after it is created, the object might be in a state called *Adding new sale items*, then a state called *Waiting for items to be shipped*, and finally, when all items have been shipped, a state called *Completed*. If you find yourself trying to use a noun to name a state, you probably have an incorrect idea about states or object classes. The name of a state should not be a noun (which is an object); it should be something that describes the object.

States are described as semipermanent conditions because external events can interrupt a state and cause the object to go to a new state. An object remains in a state until some event causes it to move, or transition, to another state. A **transition**, then, is the movement of an object from one state to another state. Transitioning is the mechanism that causes an object to leave a state and change to a new state. States are semipermanent because transitions interrupt them and

state a condition during an object's life when it satisfies some criterion, performs some action, or waits for an event

transition the movement of an object from one state to another state

FIGURE 4-25 Complete RMO CSMS domain model class diagram



© Cengage Learning®

state machine diagram a diagram showing the life of an object in states and transitions

cause them to end. Generally, transitions are short in duration—compared with states—and they cannot be interrupted. The combination of states and transitions between states provides the mechanisms that analysts use to capture business rules. In the previous RMO example, you would say that a customer sale must be in a *Paid for* state before it can transition to a *Shipped* state.

The UML diagram that is used to describe the behavior of an object is called a **state machine diagram**. Sometimes these diagrams are also called state-transition diagrams because they describe the states and the transitions of the object. This section introduces state machines.

A state machine diagram describes the behavior, or the possible behavior, of the objects in one particular object class. This is an important concept to understand—that a state machine diagram applies to only one single class, and that it can be used to describe the behavior of all objects within that class. Earlier in the chapter, you learned how to create a UML class diagram. A class diagram describes the static characteristics of objects and their associations. A state machine describes the dynamic behavior of objects.

A state machine diagram can be developed for any problem domain classes that have complex behavior or status conditions that need to be tracked. However, not all classes will require a state machine diagram. If an object in the problem domain class doesn't have status conditions that must control the processing for that object, a state machine diagram probably is not necessary. For example, in the RMO class diagram, a class such as *Sale* may need a state machine diagram. However, a class such as *SaleTransaction* probably does not. A sale transaction is created when the payment is made and then just sits there; it doesn't need to track other conditions.

A state machine diagram is composed of rounded rectangles representing the states of an object and arrows representing the transitions. **Figure 4-26** illustrates a simple state machine diagram for a printer. Because it is a little easier to learn about state machine diagrams by using tangible items, let's start with a few examples of computer hardware. The starting point of a state machine diagram is a black dot, which is called a **pseudostate**. The first shape after the black dot is the first state of the printer. In this case, the printer begins in the *Off* state.

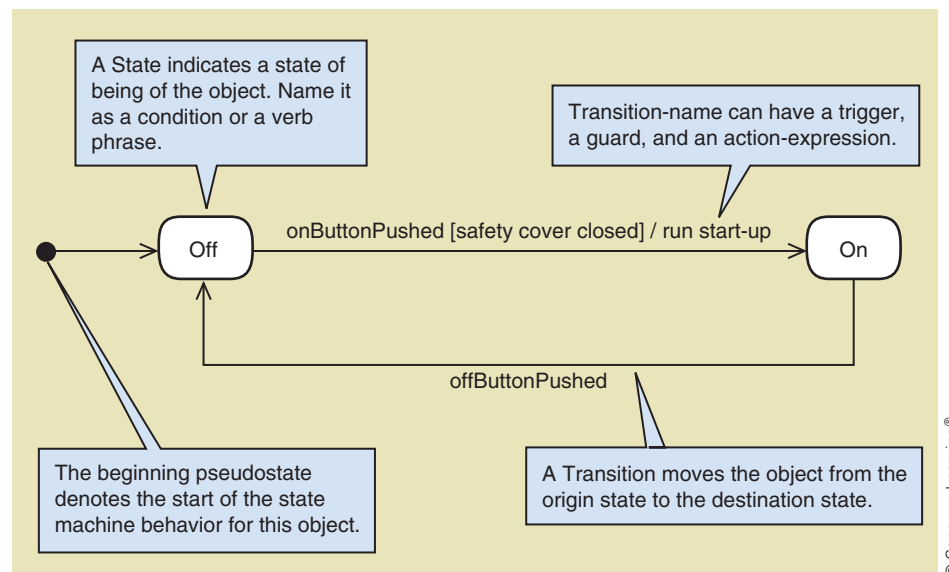
pseudostate the starting point of a state machine diagram, indicated by a black dot

destination state for a particular transition, the state to which an object moves after the completion of a transition

origin state for a particular transition, the original state of an object from which the transition occurs

As shown in Figure 4-26, the arrow leaving the *Off* state is called a *transition*. The firing of the transition causes the object to leave the *Off* state and make a transition to the *On* state. After a transition begins, it runs to completion by taking the object to the new state, called the **destination state**. A transition begins with an arrow from an **origin state**—the state prior to the transition—to a destination state, and it is labeled with a string to describe the components of the transition.

FIGURE 4-26 Simple state machine diagram for a printer



The transition label consists of the following syntax with three components:

transition-name (parameters, ...) [guard-condition] / action-expression

In Figure 4-26, the transition-name is `onButtonPushed`. The transition is like a trigger that fires or an event that occurs. The name should reflect the action of a triggering event. In Figure 4-26, no parameters are being sent to the printer so the parentheses are left off. The guard-condition is `[Safety cover closed]`. For the transition to fire, the guard must be true. The forward slash divides the firing mechanism from the actions or processes. **Action-expressions** indicate some process that must occur before the transition is completed and the object arrives in the destination state. In this case, the printer will run a start-up procedure before it goes into the *On* state.

action-expressions descriptions of the activities performed as part of a transition

guard-condition a true/false test to see whether a transition can fire

The **guard-condition** is a qualifier or test on the transition, and it is simply a true/false condition that must be satisfied before the transition can fire. For a transition to fire, first the trigger must occur and then the guard must evaluate to true. Sometimes, a transition has only a guard-condition and no triggering event. In that case, the trigger is considered to be constantly firing, and whenever the guard becomes true, the transition occurs.

Any of the three components—transition-name, guard-condition, or action-expression—may be empty. If either the transition-name or the guard-condition is empty, it automatically evaluates to true. Either of them may also be complex, with AND and OR connectives.

■ Concurrency and Concurrent States

Sometimes an object will be in two states at the same time. For example, when the printer is in the *On* state, it can also be either *Printing* or *Idle*. There are several ways to express this concurrent condition. This section explains the basic approach to modeling concurrent behavior. There are more sophisticated techniques that are available; however, the basic approach is usually sufficient for business modeling.

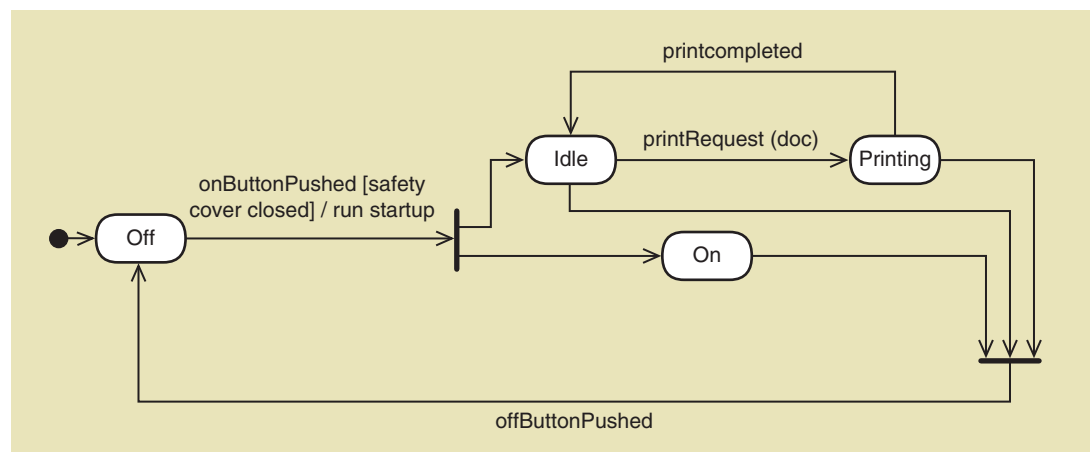
concurrency or concurrent states the condition of being in more than one state at a time

path a sequential set of connected states and transitions

This condition of being in more than one state at a time is called **concurrency**, or **concurrent states**. For example, the printer can be in the *On* state and the *Idle* state at the same time. Hence, *On* and *Idle* would be concurrent states. Another concept that is important is the idea of a path for the object. A **path** is a sequential set of connected states and transitions. In Figure 4-26, there is only one path. It is the path from the beginning state, to the *On* state then to the *Off* state, and finally looping back to the *On* state. Sometimes within a state machine diagram there will be states that are exactly concurrent states. However, when there are several states in a path that are parallel to another state, we say that those are **concurrent paths**. Figure 4-27 illustrates the example we have been describing.

concurrent paths when one or more states in a path are parallel to one or more states in another path

FIGURE 4-27 State machine diagram with concurrent paths



The notation used in Figure 4-27 is similar to the notation that you used with activity diagrams in Chapter 2. When a state has two transitions that exit that state, it is considered an OR condition. In other words, the path for the object follows only one of the transitions. To show parallel paths with the object following both paths, that is, an AND condition, a synchronization bar is used.

In Figure 4-27, the path with the *Idle* and *Printing* states is parallel with the path with the *On* state. In other words, the printer will be in either the *Idle* or *Printing* states whenever it is in the *On* state. Notice that the *printRequest(doc)* transition does not have an argument being sent with the transition. The printer will cycle between the *Idle* state and the *Printing* state whenever it is in the *On* state. When the *offButtonPushed* transition fires, the printer exits both the *On* state and either the *Idle* or *Printing* state, whichever is active. Now that you know the basic notation of state machine diagrams, you will learn how to develop a state machine diagram.

■ Rules for Developing State Machine Diagrams

State machine diagram development follows a set of rules. Usually, the primary challenge in building a state machine diagram is to identify the right states for the object. It might be helpful to pretend that you are the object itself. It is easy to pretend to be a customer, but it's a little more difficult to say "I am an order" or "I am a shipment. How do I come into existence? What states am I in?" However, if you can begin to think this way, it will help you develop state machine diagrams.

A good approach is to remember that developing state machine diagrams is an iterative behavior—more so than developing any other type of diagram. Analysts seldom get a state machine diagram right the first time. They always draw it and then refine it again and again.

Finally, don't forget to ask about an exception condition—especially when you see the words *verify* or *check*. Usually, there will be two transitions out of states that verify something: one for acceptance and one for rejection.

Here is a list of steps that will help you get started in developing state machine diagrams:

1. **Review the class diagram and select the classes that might require state machine diagrams.** Only include those classes that have multiple status conditions that are important for the system to track. Then, begin with the classes that appear to have the simplest state machine diagrams, such as the *SaleItem* class for RMO.
2. **For each selected class in the group, make a list of all the status conditions you can identify.** At this point, simply brainstorm. Remember that these states must reflect the states for the real-world objects that will be represented in software. Sometimes, it is helpful to think of the physical object, identify states of the physical object, and then translate those that are appropriate into corresponding system states or status conditions. It is also helpful to think of the life of the object. Think of activities done to the object or by the object. Often, the object will be in a particular state as these actions are occurring.
3. **Begin building state machine diagram fragments by identifying the transitions that cause an object to leave the identified state.** For example, if a sale is in a state of *Ready to be shipped*, a transition such as *beginShipping* will cause the sale to leave that state.
4. **Sequence these state-transition fragments in the correct order.** Then, aggregate these combinations into larger fragments. As the fragments are being aggregated into larger paths, it is natural to begin to look for a natural life cycle for the object.
5. **Review the paths and look for independent, concurrent paths.** When an item can be in two states concurrently, there are probably one or more concurrent paths.

6. **Look for additional transitions.** Often, during a first iteration, several of the possible combinations of state-transition-state are missed. One method to identify them is to take every paired combination of states and ask whether there is a valid transition between the states. Test for transitions in both directions.
7. **Expand each transition with the appropriate message event, guard-condition, and action-expression.** Include with each state appropriate action-expressions.
8. **Review and test each state machine diagram.** Review each of your state machine diagrams to make sure the names of the states describe the object's status condition, to make sure you have identified all the transitions, to check for all concurrent paths, and to ensure you have the exception conditions.

■ Developing RMO State Machine Diagrams

Let us practice these steps by developing two state machine diagrams for RMO. The first step is to review the domain class diagram and then select the classes that may have status conditions that need to be tracked. In this case, we select the `SaleItem` and the `InventoryItem` classes. Other classes that are candidates for state machine diagrams are: `Sale`, to track when it is completed; `Shipment`, to track arrivals at the customer location; and possibly `Customer`, to track active and inactive customers.

■ Developing the `SaleItem` State Machine Diagram

The first step in developing the `SaleItem` state machine diagram is to identify the possible status conditions that might be of interest. Some necessary status conditions are *Ready to be shipped*, *On back order*, and *Shipped*. An interesting question comes to mind at this point: Can a sale item be partially shipped? In other words, if the customer bought 10 of a single item but there are only five in inventory, should RMO ship those five and put the other five on back order? You should see the ramifications of this decision. The system and the database would need to be designed to track and monitor detailed information to support this capability. The domain class diagram for RMO indicates that a `SaleItem` can be associated with either zero (not yet shipped) shipments or one (totally shipped) shipment. Based on the current specification, the definition doesn't allow partial shipments of `SaleItems`.

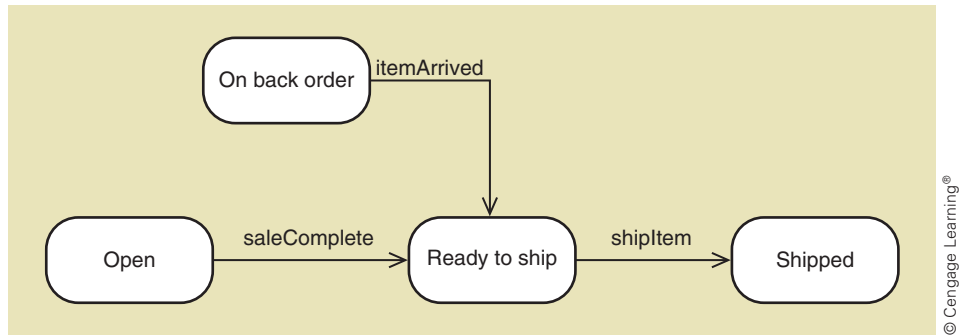
This is just another example of the benefit of building models. Had we not been developing the state machine diagram model, this question might never have been asked. The development of detailed models and diagrams is one of the most important activities that a system developer can perform. It forces analysts to ask fundamental questions. Sometimes, new system developers think that model development is a waste of time, especially for small systems. However, truly understanding the users' needs before writing the program always saves time in the long run.

The second step is to identify exit transitions for each of the status conditions. **Figure 4-28** is a table showing the states that have been defined and the exit transitions for each of those states. One additional state has been added to the list—*Open*—which covers the condition that occurs when an item has been added to the sale but the sale isn't complete or paid for, so the item isn't ready for shipping.

FIGURE 4-28 States and exit transitions for `SaleItem` object

State	Transition causing exit
Open	saleComplete
Ready to Ship	shipltem
On back order	itemArrived
Shipped	No exit transition defined

FIGURE 4-29 Partial state machine diagram for SaleItem object



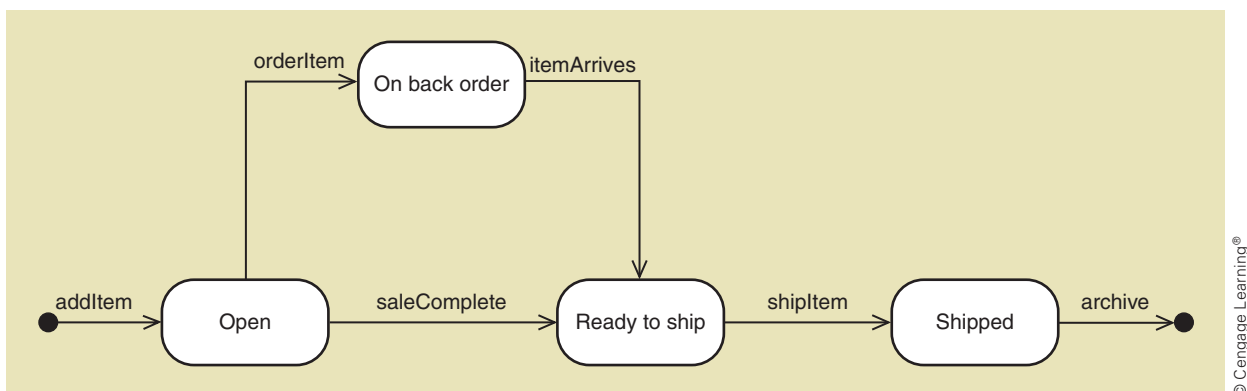
The third step is to combine the state-transition pairs into fragments and to build a state machine diagram with the states in the correct sequence. **Figure 4-29** illustrates the partially completed state machine diagram. The flow from beginning to end for the SaleItem object is quite obvious. However, at least one transition seems to be missing. There should be some path to allow entry into the *On back order* state so we recognize that this first-cut state machine diagram needs some refinement. We will fix that in a moment.

The fourth step is to look for concurrent paths. In this case, it doesn't appear that a SaleItem object can be in any two of the identified states at the same time. Of course, because we chose to begin with a simple state machine diagram, that was expected.

The fifth step is to look for additional transitions. This step is where we flesh out other necessary transitions. The first addition is to have a transition from *Open* to *On back order*. To continue, examine every pair of states to see whether there are other possible combinations. In particular, look for backward transitions. In this situation, we will not define any backward transitions.

The sixth step is to complete all the transitions with correct names, guard-conditions, and action-expressions. Two new transition-names are added. The first is the transition from the beginning black dot to the *Open* state. That transition causes the creation—or, in system terms, the instantiation—of a new SaleItem object. It is given the same name as the message into the system that adds it: *addItem*. The final transition is the one that causes the order item to be removed from the system. This transition goes from the *Shipped* state to a final circled black dot, which is a final pseudostate. On the assumption that it is archived to a backup tape when it is deleted from the active system, that transition is named *archive*. The seventh step—reviewing and testing the state machine diagram—is the quality-review step. **Figure 4-30** illustrates the final state machine diagram for SaleItem.

FIGURE 4-30 Final state machine diagram for SaleItem object



■ Developing the InventoryItem State Machine Diagram

An `InventoryItem` will have a slightly more complex state machine. In this example, you will discover the need to have some concurrent paths defined. As before, start by thinking of the various status conditions that are important and their associated end or exit conditions. As you think about an inventory item, recognize that important status conditions are associated with the level of inventory. An item can be at a normal inventory level, a low inventory level, or completely out of stock. In addition, an inventory item can be on order or have an order outstanding. It could also not have any orders outstanding. As you think about it, notice that it appears these two sets of status are independent of each other. First, identify all of these states and their exit transitions. **Figure 4-31** documents our thinking.

In the third and fourth steps, you combine these states and transitions into fragments and connect them together to give the first-cut state machine diagram. In the fifth step, you see if these state-transition fragments are really concurrent paths. **Figure 4-32** shows these steps.

There are a few things you need to consider with this first-cut state machine. Let's look at the various transitions and consider each transition along with its origin state and destination state. The top path appears to be okay—cycling between *Not on order* and *On order*. In the bottom path, however, there are transitions that have the same name. How do you ensure that the correct transitions fire and the correct states are used?

First consider the *reduceInventory* transition. The `InventoryItem` will receive this message every time an item is sold. However, it only wants to take the transition from one state to another if it is at the reorder point, or if it is the last item in stock. Let's add guards to define those conditions. You also want to initiate a reorder process when the `InventoryItem` goes to a Low stock or a Zero stock state. Let's add those action expressions.

Next consider the *restock* transition. It is correct. Depending on what state the `InventoryItem` is in, the correct transition will fire and move to the *Normal stock* state.

FIGURE 4-31 States and exit transitions for `InventoryItem` object

State	Transition causing exit
Normal stock	reduceInventory
Low stock	reduceInventory OR restock
Zero stock	removeItem OR restock
On order	itemArrives
Not on order	orderItem

FIGURE 4-32 First-cut state machine diagram for `InventoryItem` object

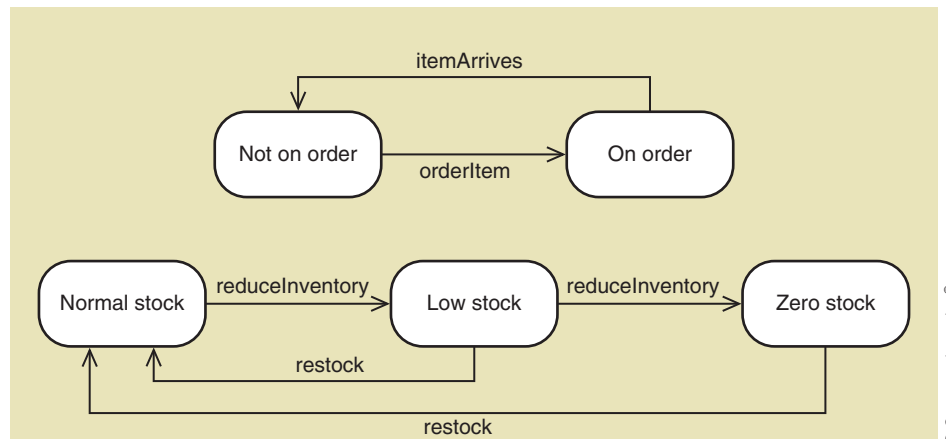
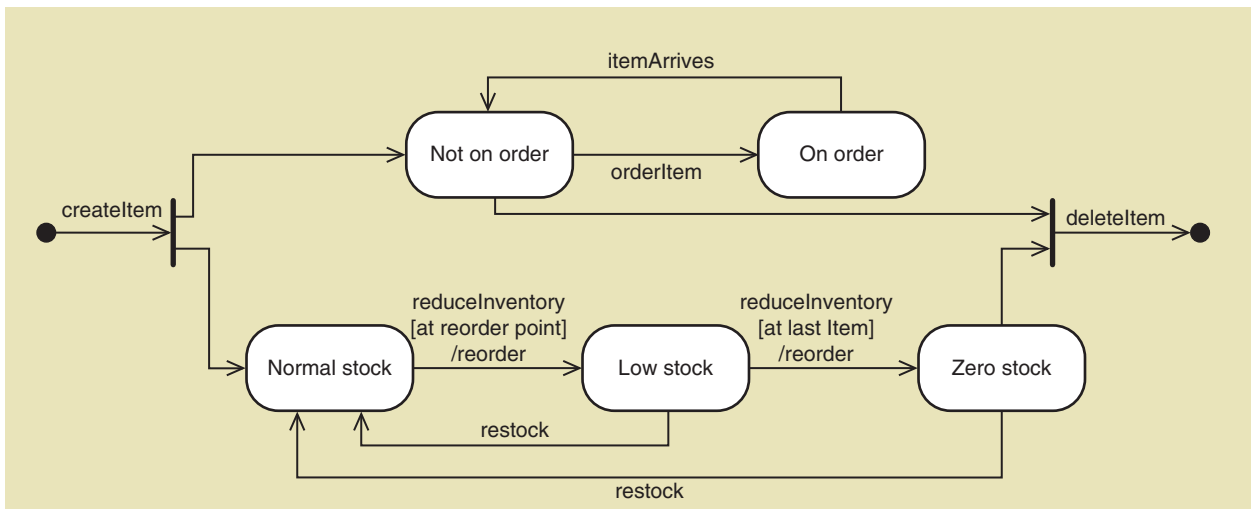


FIGURE 4-33 Final state machine diagram for InventoryItem object



© Cengage Learning®

The other thing you need to add is a starting initial state and a final state. Let's define transitions to *createItem* for the beginning and *deleteItem* for the end. *DeleteItem* is when it is removed completely from the system and the database. In other words, it is no longer offered for sale. **Figure 4-33** presents the final state machine diagram for the InventoryItem object class.

As noted by these examples, the benefit of developing a state machine diagram for a problem domain object is that it helps you capture and clarify business rules. As you develop the state machine diagrams, you must think more deeply about the behavior of these objects and what kind of conditions need to be accounted for in the program code. As always, the benefits of careful model building help you gain a true understanding of the system requirements.

CHAPTER SUMMARY

This is the second of three chapters that present techniques for modeling a system's functional requirements, highlighting the tasks that are completed during the analysis activity *Define requirements*. Use cases and things in the user's work environment are key concepts common to all approaches to system development. This chapter discusses data entities and domain classes as two terms for things in the work environment. Two techniques are demonstrated for identifying things in the problem domain: the brainstorming technique and the noun technique. The entity-relationship diagram (ERD) is used by traditional analysts and by database analysts to model things in the problem domain. An ERD shows data entities, attributes, and relationships. The UML class diagram is used for the same purpose by analysts using UML, referred to as the *domain model class diagram*. The domain model class diagram models domain classes, attributes, and associations. Multiplicity refers to the number of association

links between classes. UML and the domain model class diagram can be extended to include three types of relationships: association relationships, generalization/specialization relationships (inheritance), and whole-part relationships. Additional concepts of importance in domain model class diagrams are superclasses, subclasses, abstract classes, and concrete classes. The behavior of problem domain objects is an aspect of functional requirements that is also studied and modeled. Over time, an object will transition from one state to another. The UML state machine diagram is used to model object states and state transitions. Only complex domain classes with numerous states and state transitions will warrant a state machine diagram. Up to this point, when we talk about domain classes, they are conceptual things that users work with; they are not software classes. During design, many domain classes will become software classes and will also become tables in the relational database.

KEY TERMS

abstract class	concurrency (concurrent states)	noun technique
action-expressions	concurrent paths	origin state
aggregation	data entities	path
association	destination state	problem domain
association class	domain classes	pseudostate
attributes	domain model class diagram	relationship
binary associations	entity-relationship diagram (ERD)	semantic net
brainstorming technique	generalization/specialization relationships	state
camelback notation	guard-condition	state machine diagram
camelcase notation	identifier (key)	subclass
cardinality	inheritance	superclass
class	multiplicity	ternary association
class diagram	multiplicity constraints	transition
composition	<i>n</i> -ary association	unary association
compound attribute		whole-part relationship
concrete class		

REVIEW QUESTIONS

1. What are the two key concepts—one from Chapter 3 and one from this chapter—that define functional requirements?
2. What is the problem domain?
3. What is a “thing” called in models used by traditional analysts and database analysts?
4. What is a “thing” called in newer approaches that use UNL?
5. What are two techniques for identifying things in the problem domain?
6. What are some examples of tangible things in the problem domain of a restaurant?
7. What are some sites or locations in the problem domain of a restaurant?
8. What are some roles played by people in the problem domain of a restaurant?
9. What are the main steps of the brainstorming technique?
10. Explain why identifying nouns helps identify things in the problem domain.
11. What are the main steps of the noun technique?
12. What is an attribute, an identifier or key, and a compound attribute?
13. What is an association, and what system development standard defines it?
14. How would you describe or name the association between a ship and a captain?
15. What is the term used for association by traditional analysts and database analysts?
16. What is an association class? Why is an association class used for modeling?
17. What is multiplicity, and what is the other term used by traditional analysts and database analysts?
18. What is the minimum multiplicity for the association that reads a customer places zero or more orders?
19. What is the maximum multiplicity for the association that reads an order is placed by exactly one customer?
20. What are some examples of multiplicity constraints?
21. What are the three types of associations, and which is the most commonly used?
22. What are the three key parts of an entity-relationship diagram (ERD)?
23. Sketch a simple ERD that shows a team has zero or more players and each player is on one and only one team.
24. Sketch a semantic net that shows two teams and five players based on your ERD from question 23.

25. What is a class, a domain class, and the key parts of a class diagram?
26. What does a domain model class diagram show about system requirements, and how is it different from an ERD?
27. List appropriate UML class names by using the camelback notation for the following classes: graduate student, undergraduate major, course instructor, and final exam feedback.
28. List appropriate UML attribute names for the following attributes: student name, course grade, major name, and final exam quantity score.
29. Draw a simple domain model class diagram for the example in question #23 where a team has zero or more players and each player is on one and only one team.
30. Extend the domain model class diagram for teams and players to show a record of game statistics for each player in each game using an association class.
31. In UML, what are three types of relationships found on a class diagram?
32. What is a generalization/specialization relationship, and what object-oriented terms does it illustrate?
33. Compare/contrast superclass and subclass. Compare/contrasts abstract class and concrete class.
34. What is a whole-part relationship, and why does it show multiplicity?
35. Compare/contrast aggregation with composition for a whole-part relationship.
36. What is an object state?
37. What is a state transition?
38. When considering requirements, states and state transitions are important for understanding which other diagram?
39. What UML diagram is used to show the states and transitions for an object?
40. List the elements that make up a transition description. Which elements are optional?
41. What is a composite state? What is it used for?
42. What is meant by the term *path*?
43. What is the purpose of a guard-condition?

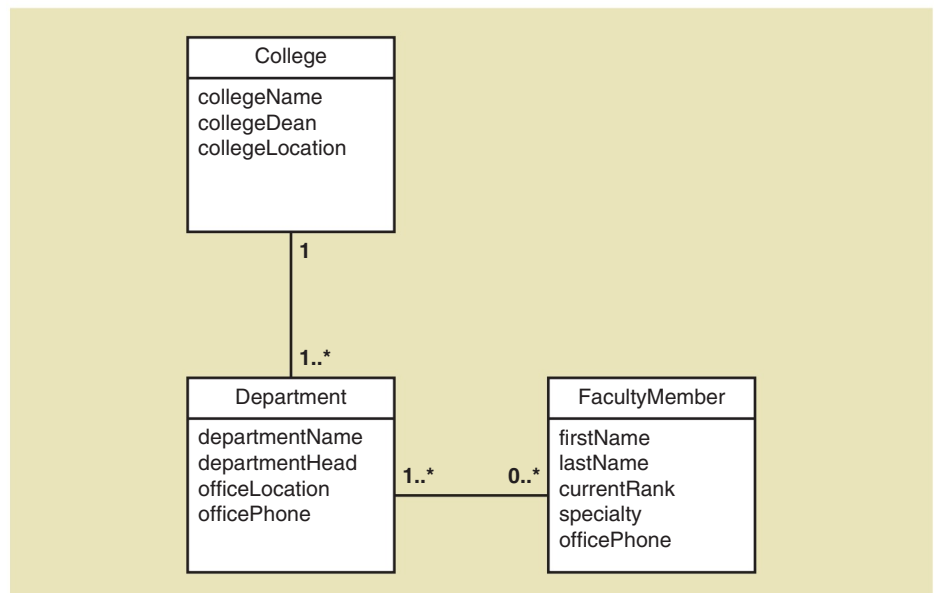
PROBLEMS AND EXERCISES

1. Draw an entity-relationship diagram, including minimum and maximum cardinality, for the following: The system stores information about two things: cars and owners. A car has attributes for make, model, and year. The owner has attributes for name and address. Assume that a car must be owned by one owner and an owner can own many cars, but an owner might not own any cars (perhaps she just sold them all, but you still want a record of her in the system).
2. Draw a class diagram for the cars and owners described in exercise 1, but include subclasses for sports car, sedan, and minivan, with appropriate attributes.
3. Consider the domain model class diagram shown in Figure 4-16—the refined diagram showing course enrollment with an association class. Does this model allow a student to enroll in more than one course section at a time? Does the model allow a course section to contain more than one student? Does the model allow a student to enroll in several sections of the same course and get a grade for each enrollment? Does the model store information about all grades earned by all students in all sections?
4. Again consider the domain model class diagram shown in Figure 4-16. Add the following to the diagram and list any assumptions you had to make: A faculty member usually teaches many course sections, but some semesters, a faculty member may not teach any. Each course section must have at least one faculty member teaching it, but sometimes, faculty teams teach course sections. Furthermore, to make sure that all course sections are similar, one faculty member is assigned as course coordinator to oversee the course, and each faculty member can be the coordinator of many courses.
5. If the domain model class diagram you drew in exercise 4 showed a many-to-many association between faculty member and course section, a further look at the association might reveal the need to store some additional information. What might this information include? (Hint: Does the instructor have specific office hours for each course section? Do you give an instructor some sort of evaluation for each course section?) Expand the domain model class diagram to allow the system to store this additional information.
6. Consider a system that needs to store information about computers in a computer lab at a university, such as the features and location of each computer. What are the domain classes that might be included in a model? What are some of the associations among these classes? What are

- some of the attributes of each class? Draw a domain model class diagram for this system.
7. Consider the domain model class diagram for the RMO CSMS Sales subsystem shown in Figure 4-20. If an `InStoreSale` is created, how many attributes does it have? If an `OnlineSale` is created, how many attributes does it have? If an existing customer places a telephone order for one item, how many new objects are created overall for this transaction? Explain.
 8. Again consider the domain model class diagram shown in Figure 4-23. How many attributes does an active cart object have? Can an on-reserve cart contain cart items? Explain.
 9. A product item for RMO is not the same as an inventory item. A product item is something like a men's leather hunting jacket supplied by Leather 'R' Us. An inventory item is a specific size and color of the jacket—like a size medium brown leather hunting jacket. If RMO adds a new jacket to its catalog and six sizes and three colors are available in inventory, how many objects need to be added overall? Explain.
 10. Consider the domain model class diagram shown in **Figure 4-34**, which includes classes for college, department, and faculty members:
 - a. What kind of UML relationships are shown in the model?
 - b. How many attributes does a faculty member have? Which (if any) have been inherited from another class?
 - c. If you add information about one college, one department, and four faculty members, how many objects do you add to the system?
 - d. Can a faculty member work in more than one department at the same time? Explain.
 - e. Can a faculty member work in two departments at the same time, where one department is in the college of business and the other department is in the college of arts and sciences? Explain.
11. Review information about your own university. Create generalization/specialization hierarchies by using the domain model class diagram notation for (a) types of faculty, (b) types of students, (c) types of courses, (d) types of financial aid, and (e) types of housing. Include attributes for the superclass and the subclasses in each case.
 12. Consider the classes involved when modeling a car and all its parts. Draw a domain model class diagram that shows the whole-part relationships involved, including multiplicity. Which type of whole-part relationships are involved?
 13. Refer to the complete RMO CSMS domain model class diagram shown in Figure 4-25. Based on that model and on the discussion of subsystems in Chapter , draw a domain model class diagram for the CSMS Marketing subsystem.
 14. Again based on the complete RMO CSMS domain model class diagram shown in Figure 4-25, draw a domain model class diagram for the CSMS Order Fulfillment subsystem.
 15. Based on the following description of a shipment made by Union Parcel Shipments, identify all the states and exit transitions and then develop a state machine diagram.

A shipment is first recognized after it has been picked up from a customer. Once in the system,

FIGURE 4-34 Domain model class diagram for a university



it is considered active and in transit. Every time it goes through a checkpoint, such as arrival at an intermediate destination, it is scanned and a record is created indicating the time and place of the checkpoint scan. The status changes when it is placed on the delivery truck. It is still active, but now it is also considered to have a status of *delivery pending*. After it is delivered, the status changes again.

From time to time, a shipment has a destination that is outside the area served by Union. In those cases, Union has working relationships with other courier services. After a package

is handed off to another courier, it is noted as being handed over. In those instances, a tracking number for the new courier is recorded (if it is provided). Union also asks the new courier to provide a status change notice after the package has been delivered.

Unfortunately, from time to time, a package gets lost. In that case, it remains in an active state for two weeks but is also marked as misplaced. If after two weeks the package hasn't been found, it is considered lost. At that point, the customer can initiate lost-package procedures to recover any damages.

CASE STUDY

Metropolitan Car Service Bureau

Metropolitan Car Service Bureau needs a system that keeps car service records. The company's analyst has provided information about the problem domain in the form of notes. Your job is to use those notes to draw the domain model class diagram. The analyst's notes are as follows:

- The Owner class has the attributes name and address.
- The Vehicle class is an abstract class that has the attributes VIN, model, and model year.
- There are two types of vehicles, cars and trucks:
 - Car has additional attributes for the number of doors and luxury level.
 - Truck has an additional attribute of cargo capacity.
- The Manufacturer class has the attributes name and location.
- The Dealer class has the attributes name and address.

A service record is an association class between each vehicle and a dealer, with the attributes service date and current mileage. A warranty service record is a special type of service record with an additional attribute: eligibility verification. Each service record is associated with a predefined service type, with the attributes type ID, description, and labor cost. Each service type is associated with zero or more parts, with the attributes part ID, description, and unit cost. Parts are used with one or more service types.

An owner can own many vehicles, and a vehicle can be owned by many owners. An owner and a vehicle are entered into the system only when an owned vehicle is first serviced by a dealer. Vehicles are serviced many times at various dealers, which service many vehicles.

1. Draw a UML domain model class diagram for the system as described here. Be as specific and accurate as possible, given the information provided. If needed information is not given, make realistic assumptions.
2. Answer True or False to the following statements, which are based on the domain model. You may want to draw a semantic net to help you think through the questions.
 - a. This domain model is for a single car dealer service department.
 - b. This domain model is for a single car manufacturer.
 - c. A vehicle can have service records with more than one dealer.
 - d. A dealer can service vehicles from more than one manufacturer.
 - e. Current mileage is recorded for service records and warranty service records.
 - f. An owner can have each of his or her cars serviced by a different dealer.
 - g. A warranty service for a car can include many parts.
 - h. A vehicle can be made by more than one manufacturer.
3. Consider that a vehicle goes through many states and state transitions from the perspective of Metropolitan. For example, a new vehicle might be brought in for the first time. A previously serviced vehicle might be brought in. Think through the sequences that go on for a vehicle when it is being worked on by Metropolitan. Draw a state machine diagram showing states and state transitions, including names for the transitions.

RUNNING CASE STUDIES

Community Board of Realtors®

In Chapter 3, you identified use cases for the Board of Realtors Multiple Listing Service (MLS) system, which supplies information that local real estate agents use to help them sell houses to their customers. During the month, agents list houses for sale (listings) by contracting with homeowners. Each agent works for a real estate office, which sends information on listings to the Multiple Listing Service. Therefore, any agent in the community can get information on the listing. Much of the information is available to potential customers on the Internet.

Information on a listing includes the address, year built, square feet, number of bedrooms, number of bathrooms, owner name, owner phone number, asking price, and status code. Additionally, many pictures and videos showing features of the listing are included. It is also important to have information on the listing agent, such as name, office phone, cell phone, and e-mail address. Agents work through a real estate office, so it is important to know the office name, office manager name, office phone, and street address.

1. Based on the information here, draw a domain model class diagram for the MLS system. Be sure to consider what information needs to be included versus information that is not in the problem domain. For example, is detailed information about the owner, such as his employer or his credit history, required in the MLS system?

Is that information required regarding a potential buyer?

2. Draw a second domain model class diagram that adds the following specifications. First, there are two types of listings: a listing for sale and a listing for lease. Additionally, a listing might include no structures, such as vacant land, or it might include more than one structure, such as a main house and a guest house, each with separate values for square footage, number of bedrooms, and number of bathrooms.
3. Draw a third domain model class diagram that assumes a listing might have multiple owners. Additionally, a listing might be shared by two or more agents, and the percentage of the commission that each agent gets from the sale can be different for each agent.
4. Consider that a real estate listing goes through several different states over time. For example, it might be a new listing, a mature listing, a revised listing, an under contract listing, and a sold listing. Draw a state machine diagram for a listing object based on this information. Include transition names and be sure to consider all of the possible transitions. For example, can a new listing transition directly to an under contract listing?

The Spring Breaks 'R' Us Travel Service

In Chapter 3, you identified use cases for the Social Networking subsystem SBRU is researching. Let us assume you were thinking about a number of potential domain classes that might be involved. For example, there would need to be information about a traveler attending a resort for a particular week. The traveler would be assigned to a room along with roommates but might also be connected to other friends. There might be different interests or hobbies a traveler can associate with in the hopes of connecting to others. The resort has many locations where a traveler might be hanging out at any given time, and a traveler can note whether the location is “liked.”

People might schedule a party at a location and invite specific friends.

1. For the Social Networking subsystem as described here, list the domain classes and their attributes that should be included in the Social Networking subsystem. Be creative and add those you think should be included to make the system useful and appealing.
2. Based on the domain classes you identified, draw a domain model class diagram showing domain classes with attributes and associations with multiplicity.

On the Spot Courier Services

On the Spot courier services grew and changed over the years. At first, Bill received requests for package pickups on his mobile phone, recorded that information in a log, and would then drive around to retrieve all the packages later in the day. However, he soon discovered that with another driver, it was difficult to coordinate pickups between the two of them from his van. It was not long before he reorganized his business and turned the warehouse employee into a driver. Then, he stayed in the warehouse himself, and his two employees made all the pickups and deliveries. This worked well because he could control and coordinate the pickups and deliveries better. It was also easier for him to receive pickup requests working at a desk rather than trying to do it while driving a delivery van.

As he thought about how his business was growing and the services he provided to his customers, Bill began to itemize the kinds of information he would need to maintain.

Of course, he needed to maintain information about his customers. Some of his customers were businesses; some were individuals. He needed to have basic address and contact information for every customer. Also, for his corporate customers, he needed to identify a primary contact person. It was mostly his corporate customers who wanted to receive monthly statements listing all their shipments during the month and the total cost. Bill needed to distinguish which customers paid cash and which wanted monthly statements. In fact, for those who paid monthly, he needed to keep a running account of such things as when they were last billed, when they paid, and any outstanding balances. Finally, when payments were received, either for individual shipments or from monthly invoices, he needed to record information about the payment: type of payment, date, and amount. Although this was not a sophisticated billing and payment system, Bill thought it would suffice for his needs.

Next, he started thinking about his packages and shipments. At the time that a request for a pickup came in, he needed to keep track of it as some type of delivery request or delivery order. At that point in time, Bill mostly needed to know who the customer

was, where the pickup location was, and what date and time the package(s) would be ready for pickup. He also recorded the date and time that he received the order. A delivery order was considered “open” until the delivery van arrived at the pickup location and the packages were all retrieved. At that point, the delivery order was satisfied.

Once the packages were retrieved, each package needed to be uniquely identified. Bill needed to know when it was picked up and which delivery person picked it up. Other important information was the “deliver to” entity name and the address. He also needed to identify the type of delivery. Some packages were high priority, requiring same-day delivery. Others were overnight. Of course, the weight and cost were recorded so the customer could either pay or have it added to the monthly invoice.

In the courier and delivery business, one of the most important information requirements is the date and time stamp. For each package, it is important to know when it was picked up, when it arrived at the warehouse, when it went back out on the delivery run, and when it was delivered. When possible, it is also important to have names associated with each of these events.

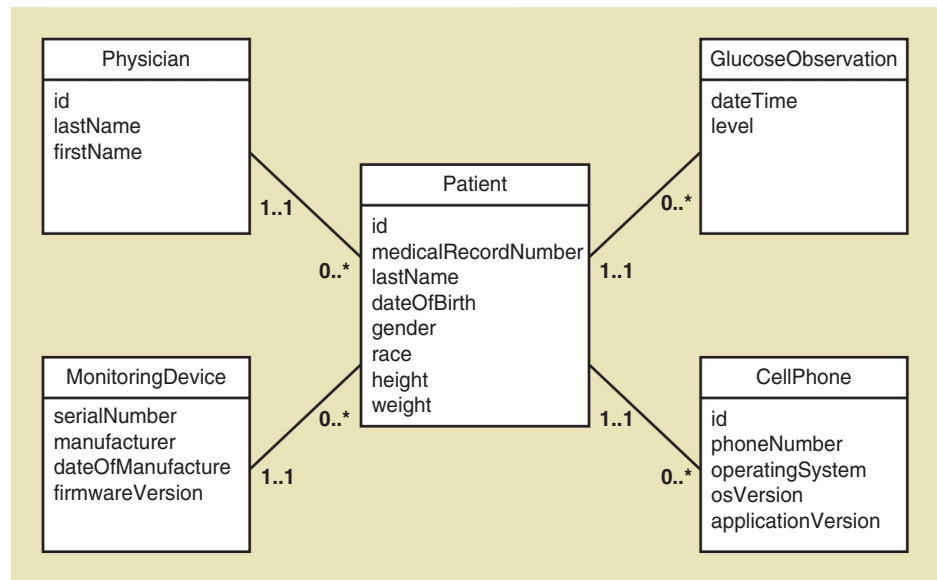
1. Using the noun technique, read through this case and identify all the nouns that may be important for this system. You may also find it helpful to read back through the case descriptions in the previous chapters.
2. Once you have identified all the nouns, identify which are classes and which are attributes of these primary classes. Begin constructing a class diagram based on the classes and attributes you have identified.
3. Now that you have identified the classes, determine what the relationships should be among the classes. Add multiplicity constraints, being especially cognizant of zero-to-many versus one-to-many differences.
4. Finalize the class diagram, including all your classes, attributes, primary keys, relationships, and multiplicity constraints.

Sandia Medical Devices

Initial discussions about the functional requirements resulted in an initial domain model class diagram for Sandia Medical Devices’ Real-Time Glucose Monitoring (RTGM) system (see **Figure 4-35**). After consultations with system stakeholders, the following potential changes to the diagram are being considered:

- Include additional medical personnel (nurses and physicians’ assistants, at a minimum).
- Include alerts sent by the system to medical personnel and messages sent by medical personnel to the patient.

FIGURE 4-35 Initial domain model class diagram for Sandia RTGM system



1. Modify the diagram to incorporate the changes under consideration. You may need to use generalization/specialization (inheritance).
2. Are a set of abstract and concrete classes needed to represent variations among cell phones? Why or why not?

FURTHER RESOURCES

- Classic and more recent texts include the following:
- Grady Booch, Ivar Jacobson, and James Rumbaugh, *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.
 - Craig Larman, *Applying UML and Patterns* (3rd ed.). Prentice Hall, 2005.
 - Peter Rob and Carlos Coronel, *Database Systems: Design, Implementation, and Management* (7th ed.). Course Technology, 2007.

CHAPTER FIVE

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- ▶ Write fully developed use case descriptions
- ▶ Develop activity diagrams to model flow of activities
- ▶ Develop system sequence diagrams
- ▶ Use the CRUD technique to validate use cases
- ▶ Explain how use case descriptions and UML diagrams work together to define functional requirements

CHAPTER OUTLINE

- ▶ Use Case Descriptions
- ▶ Activity Diagrams for Use Cases
- ▶ The System Sequence Diagram—Identifying Use Case Inputs and Outputs
- ▶ SSD Notation
- ▶ Use Cases and CRUD
- ▶ Integrating Requirements Models

OPENING CASE ELECTRONICS UNLIMITED: INTEGRATING THE SUPPLY CHAIN

Electronics Unlimited is a warehousing distributor that buys electronic equipment from various suppliers and sells it to retailers throughout the United States and Canada. It has operations and warehouses in Los Angeles, Houston, Baltimore, Atlanta, New York, Denver, and Minneapolis. Its customers range from large nationwide retailers, such as Target, to medium-sized independent electronics stores.

Most large retailers have moved toward integrated supply chains. Information systems used to be focused on processing internal data; however, today, these retail chains want suppliers to become part of a totally integrated supply chain system. In other words, the systems need to communicate between companies to make the supply chain more efficient.

To maintain its position as a leading wholesale distributor, Electronics Unlimited has to convert its system to link with its suppliers (the manufacturers of the electronic equipment) and its customers (the retailers). It is developing a completely new system that uses object-oriented techniques to provide these links. Object-oriented techniques facilitate system-to-system interfaces by using predefined components and objects to accelerate the development process. Fortunately, many of the system development staff members have experience with object-oriented development and are eager to apply the techniques and models to the system development project.

William Jones is explaining object-oriented development to the group of systems analysts who are being trained in this approach.

“We’re developing most of our new systems by using object-oriented principles,” he tells them. “The complexity of the new system, along with its interactivity, makes the object-oriented approach a natural way to develop requirements. The object-oriented models track

very closely with the new object-oriented programming languages and frameworks.”

William is just getting warmed up.

“This way of thinking about a system in terms of objects is very interesting,” he adds. “It is also consistent with the object-oriented programming techniques you learned in your programming classes. You probably first learned to think about objects when you developed screens for the user interface. All the controls on the screen, such as buttons, text boxes, and drop-down boxes, are objects. Each has its own set of trigger events that activate its program functions.”

“How does this apply to our situation?” one of the analysts asks.

“You just extend that thought process,” William explains. “You think of such things as purchase orders and employees as objects, too. We can call them the problem domain objects to differentiate them from user-interface objects, such as windows and buttons. During analysis, we have to find out all the trigger events and methods associated with each business object.”

“And how do we do that?” another analyst asks.

“You continue with your fact-finding activities and build a better understanding of each use case,” William says. “The way the problem domain objects interact with each other in the use case determines how you identify the initiating activity. We refer to those activities as the messages between objects. The tricky part is that you need to think in terms of objects instead of just processes. Sometimes, it helps me to pretend I am an object. I will say, ‘I am a purchase order object. What functions and services are other objects going to ask me to do?’ After you get the hang of it, it works very well, and it is enlightening to see how the system requirements unfold as you develop the diagrams.”

■ Overview

The main objective of defining requirements in system development is understanding users’ needs, how the business processes are carried out, and how the system will be used to support those business processes. As indicated in Chapter 2, system developers use a set of models to discover and understand the requirements for a new system. This activity is a key part of systems analysis in the system development process. The first step in the process for developing this understanding requires the fact-finding skills you learned in Chapter 2. Fact-finding activities are also called *discovery activities*, and obviously, discovery must precede understanding.

The models introduced in Chapters 3 and 4 focus on two primary aspects of functional requirements: the use cases and the problem domain classes involved in users’ work. User stories are sometimes used in place of use cases with Agile development. Use cases are identified by using the user goal technique and the

event decomposition technique. The Unified Modeling Language (UML) use case diagram was introduced to show use cases and actors. An information system also needs to record and store information about classes of objects involved in the business processes. In a manual system, the information is recorded on paper and stored in a filing cabinet. In an automated system, the information is stored in electronic files or a database. The information storage requirements of a system are documented either with entity-relationship diagrams (ERDs) or, as emphasized in this text, with UML domain model class diagrams.

In this chapter, you learn additional techniques and models that will allow you to extend the requirements models to show additional information about the use cases for the system. Fully developed use case descriptions, UML activity diagrams, and UML system sequence diagrams (SSDs) are introduced to show more information about each use case. Additionally, the CRUD technique is introduced to help validate the use cases in terms of supporting all of the domain classes in the domain model class diagram. Remember, when defining requirements for a system, you will also be doing design and implementation work, as illustrated in the Trade Show application developed in Chapter 1.

■ Use Case Descriptions

use case description a textual model that lists and describes the processing details for a use case

A list of use cases and use case diagrams provides an overview of all the use cases for a system. Detailed information about each use case is described with a **use case description**. Brief use case descriptions were introduced in Chapter 3. This chapter introduces fully developed use case descriptions that list and describe the processing details for a use case.

■ Brief Use Case Descriptions

Depending on an analyst's needs, use case descriptions tend to be written at two separate levels of detail: brief description and fully developed description. Some brief use case descriptions were shown in Chapter 3 (repeated again as **Figure 5-1**). A brief description gives enough detail for very simple use cases, especially when the system to be developed is a small, well-understood application. Examples of simple use cases are *Add product comment* or *Send message*. A use case such as *Fill shopping cart* is complex enough that a fully developed description is also written after the initial brief use case description is finalized.

■ Fully Developed Use Case Descriptions

The fully developed description is the most formal method for documenting a use case. One of the major difficulties for software developers is that they often struggle to obtain a deep understanding of the users' needs. But if you create a fully developed use case description, you increase the probability that you thoroughly understand the business processes and the ways the system must support them. To create a comprehensive, robust system that truly meets users' needs,

FIGURE 5-1 Use cases and brief use case descriptions

Use case	Brief use case description
Create customer account	User/actor enters new customer account data, and the system assigns account number, creates a customer record, and creates an account record.
Look up customer	User/actor enters customer account number, and the system retrieves and displays customer and account data.
Process account adjustment	User/actor enters order number, and the system retrieves customer and order data; actor enters adjustment amount, and the system creates a transaction record for the adjustment.

FIGURE 5-2 Fully developed use case description for Create customer account

Use case name:	<i>Create customer account.</i>	
Scenario:	Create online customer account.	
Triggering event:	New customer wants to set up account online.	
Brief description:	Online customer creates customer account by entering basic information and then following up with one or more addresses and a credit or debit card.	
Actors:	Customer.	
Related use cases:	Might be invoked by the <i>Check out shopping cart</i> use case.	
Stakeholders:	Accounting, Marketing, Sales.	
Preconditions:	Customer Account subsystem must be available. Credit/debit authorization services must be available.	
Postconditions:	Customer must be created and saved. One or more Addresses must be created and saved. Credit/debit card information must be validated. Account must be created and saved. Address and Account must be associated with Customer.	
Flow of activities:	Actor	System
	1. Customer indicates desire to create customer account and enters basic customer information.	1.1 System creates a new customer. 1.2 System prompts for customer addresses.
	2. Customer enters one or more addresses.	2.1 System creates addresses. 2.2 System prompts for credit/debit card.
	3. Customer enters credit/debit card information.	3.1 System creates account. 3.2 System verifies authorization for credit/debit card. 3.3 System associates customer, address, and account. 3.4 System returns valid customer account details.
Exception conditions:	1.1 Basic customer data are incomplete. 2.1 The address isn't valid. 3.2 Credit/debit information isn't valid.	

© Cengage Learning®

you must understand the detailed steps of each use case. Internally, a use case includes a whole sequence of steps to complete a business process. **Figure 5-2** is an example of a fully developed use case description of the use case *Create customer account*.

Figure 5-2 also serves as a standard template for documenting a fully developed description for other use cases. The first and second compartments are used to identify the use case and the specific scenario within the use case (if needed) that is being documented. Frequently, several variations of the business steps exist within a single use case. These different flows of activities are called **scenarios** or sometimes **use case instances**. The use case *Create customer account* will have a separate flow of activities depending on which actor invokes the use case. The processes for a customer service representative updating information over the phone might be quite different from the processes for a customer updating the information him or herself. Figure 5-2 shows an online customer creating a customer account, so the scenario indicated is *Create online customer account*. Another use case description would be written for the *Create*

scenarios or use case instances
a unique set of internal activities within a use case

customer account by phone scenario. Each flow of activities is a valid sequence for the *Create customer account* use case. Thus, a scenario is a unique set of internal activities within a use case and represents a unique path through the use case.

In larger or more formal projects, a unique identifier can also be added for the use case, with an extension identifying the particular scenario. Sometimes, the name of the system developer who produced the form is added.

The third compartment identifies the event that triggers the use case. The fourth compartment is a brief description of the use case. Analysts may just duplicate the brief description they constructed earlier here. The fifth compartment identifies the actor or actors. Implied in all use cases is a person who uses the system called an actor. Actors are shown as stick figures on use case diagrams. An actor is always outside the automation boundary of the system but may be part of the manual portion of the system. By defining actors that way—as those who interact with the automated part of the system—you can more precisely define the exact interactions to which the automated system must respond.

The sixth compartment identifies other use cases and the way they are related to this use case. These cross-references to other use cases help document all aspects of the users' requirements.

The seventh compartment identifies stakeholders who are interested parties other than specific actors. They might be users who don't actually invoke the use case but who have an interest in results produced from the use case. For example, in Figure 5-2, the Accounting Department is interested in accurately capturing billing and credit card information. Although no one in the Marketing Department actually creates new customer accounts, they do perform statistical analysis of the new customers and create marketing promotions. Thus, marketers have an interest in the data that are captured and stored from the *Create customer account* use case. The Sales Department is interested in having an easy-to-use and attractive user interface to ensure sales aren't lost because of poor user experience. Considering all the stakeholders is important for system developers so they ensure they have understood all requirements.

The eighth and ninth compartments—preconditions and postconditions—provide critical information about the state of the system before and after the use case executes. **Preconditions** identify what the state of the system must be for the use case to begin, including what objects must already exist, what information must be available, and even the condition of the actor prior to beginning the use case.

preconditions conditions that must be true before a use case begins

postconditions what must be true upon the successful completion of a use case

Postconditions identify what must be true upon completion of the use case. Most important, they indicate what new objects are created or updated by the use case and how objects need to be associated. The postconditions are important for two reasons. First, they form the basis for stating the expected results for test cases that will be used for testing the use case after it is implemented. For example, in the *Create customer account* use case, it is important to test that a customer record, address record, and account record were successfully added to the database. Second, the objects in postconditions indicate which objects involved in the use case are important for design. You will see in Chapters 12 and 13 that the design of a use case includes identifying and assigning responsibilities to objects that collaborate to complete the use case. In this situation, a customer, one or more addresses, and an account object collaborate to create a new customer account.

The tenth compartment in the template describes the detailed flow of activities of the use case. In this instance, Figure 5-2 shows a two-column version, identifying the steps performed by the actor and the responses required by the system. The item numbers help identify the sequence of the steps. Alternative activities and exception conditions are described in the eleventh compartment. The numbering of exception conditions also helps tie the exceptions to specific steps in the flow of activities.

FIGURE 5-3 Fully developed use case description for Ship items

Use case name:	<i>Ship items.</i>	
Scenario:	Ship items for a new sale.	
Triggering event:	Shipping is notified of a new sale to be shipped.	
Brief description:	Shipping retrieves sale details, finds each item and records it is shipped, records which items are not available, and sends shipment.	
Actors:	Shipping clerk.	
Related use cases	None.	
Stakeholders:	Sales, Marketing, Shipping, warehouse manager.	
Preconditions:	Customer and address must exist. Sale must exist. Sale items must exist.	
Postconditions:	Shipment is created and associated with shipper. Shipped sale items are updated as shipped and associated with the shipment. Unshipped items are marked as on back order. Shipping label is verified and produced.	
Flow of activities:	Actor	System
	1. Shipping requests sale and sale item information.	1.1 System looks up sale and returns customer, address, sale, and sales item information.
	2. Shipping assigns shipper.	2.1 System creates shipment and associates it with the shipper.
	3. For each available item, shipping records item is shipped.	3.1 System updates sale item as shipped and associates it with shipment.
	4. For each unavailable item, shipping records back order.	4.1 System updates sale item as on back order.
	5. Shipping requests shipping label supplying package size and weight.	5.1 System produces shipping label for shipment. 5.2 System records shipment cost.
Exception conditions:	2.1 Shipper is not available to that location, so select another. 3.1 If order item is damaged, get new item and updated item quantity. 3.1 If item bar code isn't scanning, shipping must enter bar code manually. 5.1 If printing label isn't printing correctly, the label must be addressed manually.	

© CengageLearning®

Figure 5-3 shows the use case description for the RMO use case *Ship items*. The scenario for this description assumes they are shipping a new sale rather than back-ordered items from a previous sale. Notice that the use case description minimizes the description of manual work that is done in conjunction with shipping items. Some analysts put that detail in, but others don't because the emphasis is on the interaction with the automated part of the application. In this use case, the preconditions show what existing objects must already exist before the use case can execute. They can't ship items that aren't part of an existing sale for a customer. The preconditions show that this use case will involve a customer, sale, and sale items objects. The postconditions again indicate what to look for when stating the expected results for a test case and show the objects that will need to collaborate in the design. This use case will also involve a shipper, shipment, shipped items, and back-ordered items.

■ Activity Diagrams for Use Cases

Another way to document a use case is with a UML activity diagram. In Chapter 2, you learned about activity diagrams as a form of workflow diagram that might cover several use cases. Activity diagrams are also used to document the flow of activities for one use case.

Figure 5-4 is an activity diagram that documents the flow of activities for the *Create customer account* use case. Sometimes, an activity diagram can take the place of the flow of activities section of a use case description, and sometimes, it is created to supplement the use case description. In this example, there are two swimlanes: one for the customer and one for the system. The customer has three activities, and the system has five activities.

An example is shown in **Figure 5-5** for the *Ship items* use case previously seen in Figure 5-3. One of the strengths of activity diagrams is that it provides a more graphical view of the flow of activities. Figure 5-5 illustrates both a repeating set of steps, that is for each *SaleItem* in the *Sale*, and a decision point to choose which set of steps to perform. Even though this flow is described by the use case description, it is more evident in the activity diagram. Figure 5-5 illustrates a correct use of the beginning and ending synchronization bars and the decision diamond. Remember that synchronization

FIGURE 5-4 Activity diagram for Create customer account showing alternate way to model the flow of activities

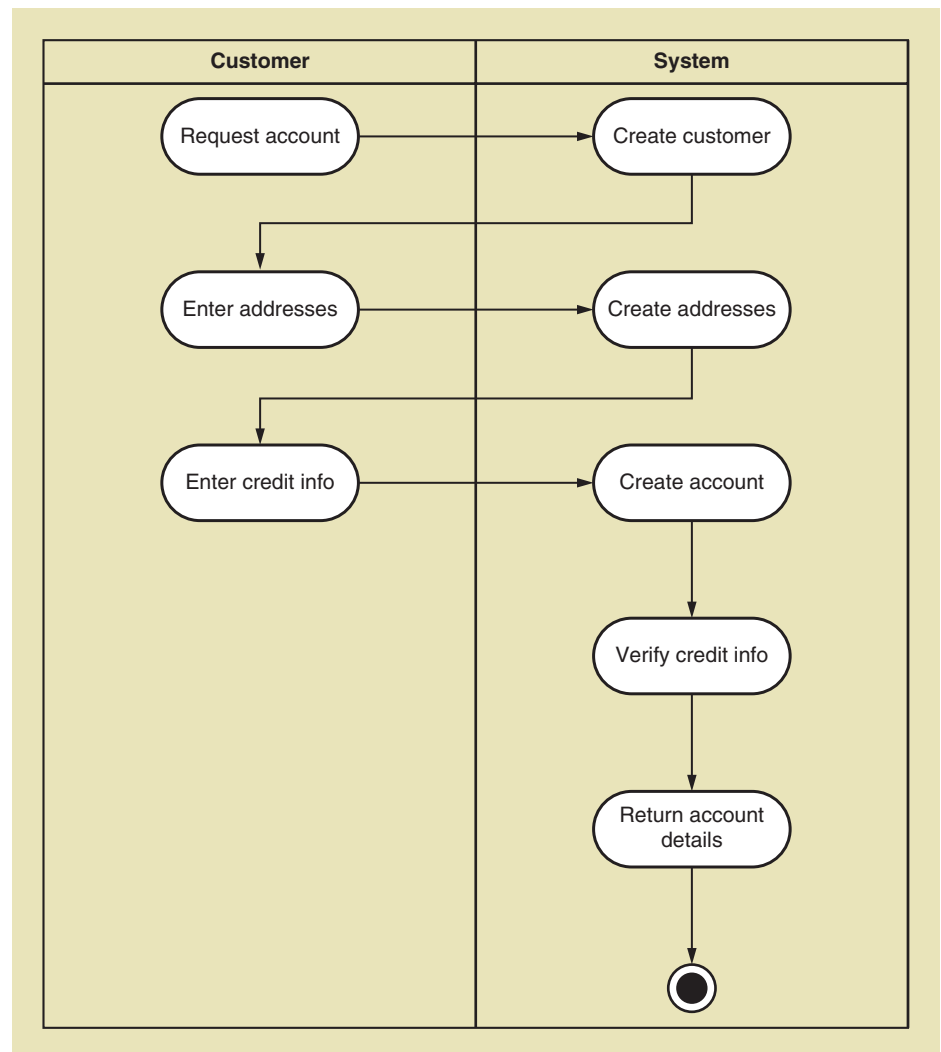
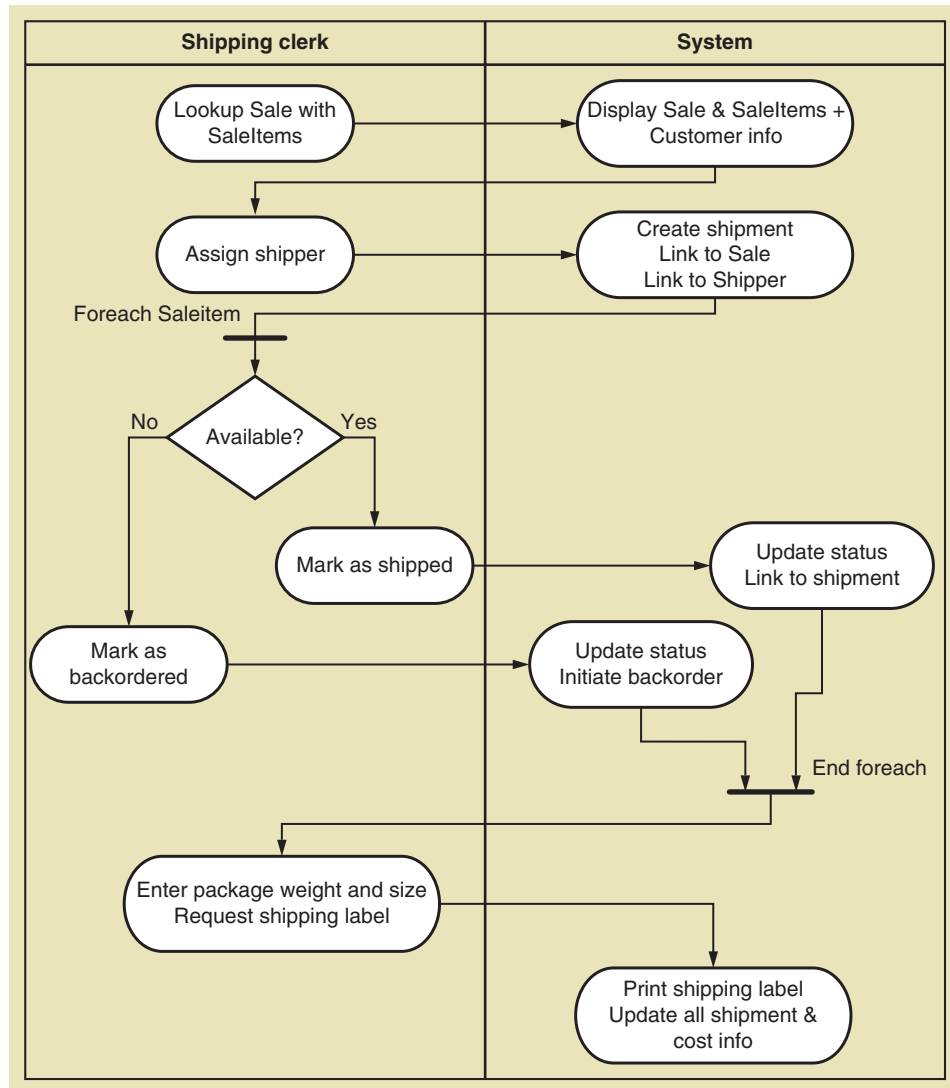


FIGURE 5-5 Activity diagram for Ship Items use case

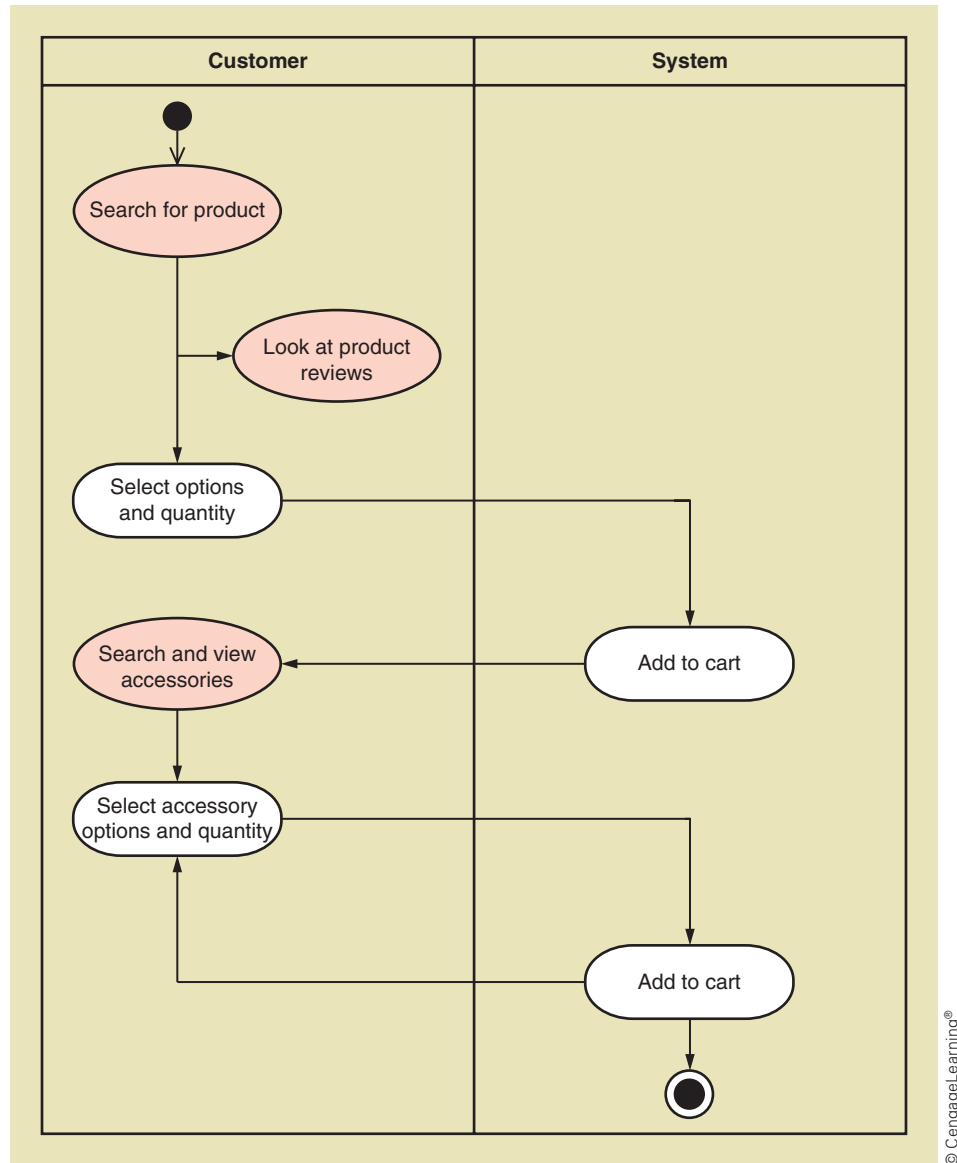


© Cengage Learning®

bars can be used either for parallel concurrent paths, or for beginning and ending loops. In this example, we see a loop and inside the loop a decision point to initiate independent paths, which are later rejoined at the ending synchronization bar.

Activity diagrams are helpful when the flow of activities for a use case is complex. The use case *Fill shopping cart* is complex in that three other use cases might be invoked while adding items to the shopping cart. For example, the actor might search for a product and then look at product reviews before adding the item to the cart. Once an item is added, the actor might search for and view available accessories and then add one or more to the cart. The activity diagram shown in Figure 5-6 shows the *Fill shopping cart* use case flow of activities. The shaded ovals show the other use cases that are invoked while filling the shopping cart. The activities of *Fill shopping cart* go in between the other use cases. For example, after invoking *Search for product* and then *Look at product reviews*, the actor might start *Fill shopping cart* to select options and quantities and add it to the cart. Then the actor might switch to *Search and view accessories* before continuing *Fill shopping cart* to add an accessory. The activity diagram can be used to show a richer user experience in this way.

FIGURE 5-6 Activity diagram for Fill shopping cart showing richer user experience



■ The System Sequence Diagram—Identifying Inputs and Outputs

system sequence diagram (SSD)

a diagram showing the sequence of messages between an actor and the automated part of the system during a use case or scenario

In the object-oriented approach, the flow of information is achieved through sending messages either to and from actors or back and forth between internal objects. A **system sequence diagram (SSD)** is used to describe this flow of information into and out of the automated portion of the system. Thus, an SSD documents the inputs and the outputs and identifies the interaction between actors and the system. It is an effective tool to help in the initial design of the user interface by identifying the specific information that flows from the user into the system and the information that flows out of the system back to the user. An SSD is a special type of UML sequence diagram. You will learn more about detailed sequence diagrams in Chapter 13.

■ SSD Notation

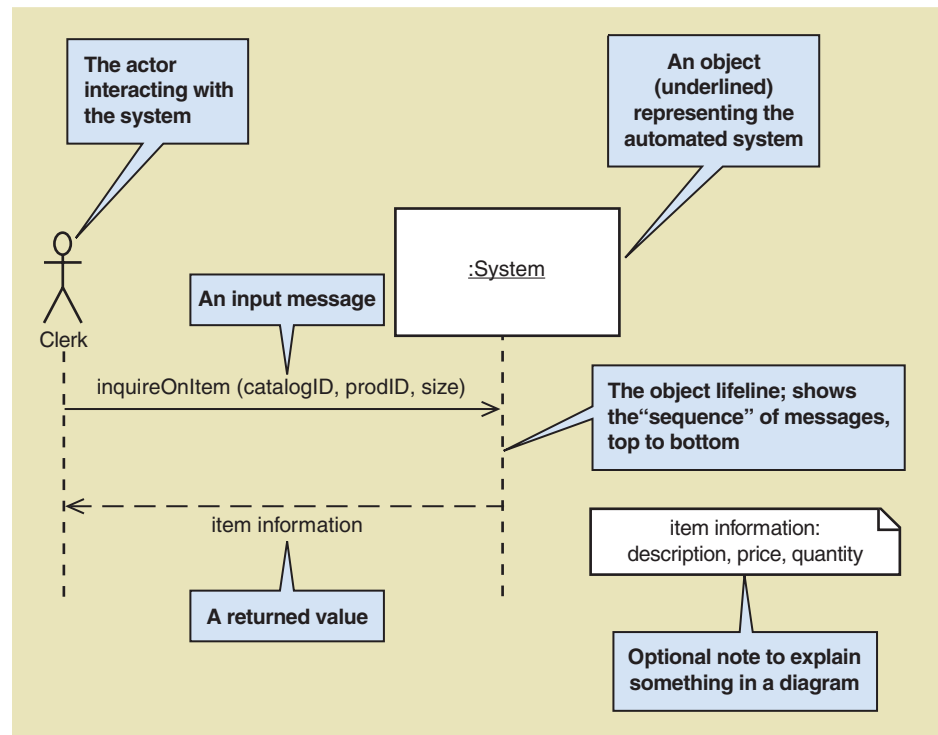
Figure 5-7 shows a generic SSD with callouts annotating the diagram. As with a use case diagram, the stick figure represents an actor—a person (or role) who interacts with the system. In a use case diagram, the actor “uses” the system, but the emphasis in an SSD is on how the actor “interacts” with the system by entering input data and receiving output data. The box labeled :System is an object that represents the entire automated system. In SSDs and all other interaction diagrams, analysts use object notation instead of class notation. In object notation, a box refers to an individual object, not the class of all similar objects. The notation is simply a rectangle with the name of the object underlined. The colon before the underlined class name is a frequently used but optional part of the object notation to indicate that the object is an unnamed object of the class. In an SSD, the only object included is one representing the entire system: an unnamed object of the System class.

lifeline, or **object lifeline** the vertical line under an object on a sequence diagram to show the passage of time for the object

Underneath the actor and :System are vertical dashed lines called *lifelines*. A **lifeline**, or **object lifeline**, is simply the extension of that object—either actor or object—during the use case. The arrows between the lifelines represent the messages that are sent by the actor. Each arrow has an origin and a destination. The origin of the message is the actor or object that sends it, as indicated by the lifeline at the arrow’s tail. Similarly, the destination actor or object of a message is indicated by the lifeline that is touched by the arrowhead. The purpose of lifelines is to indicate the sequence of the messages sent and received by the actor and object. The sequence of messages is read from top to bottom in the diagram.

A message is labeled to describe its purpose and any input data being sent. The message name should follow the verb-noun syntax to make the purpose clear. The syntax of the message label has several options; the simplest forms are shown in Figure 5-7. Remember that the arrows are used to represent a message and input data. But what is meant by the term *message* here? In a sequence diagram, a message is an action that is invoked on the destination object, much like a command. Notice in Figure 5-7 that the input message is called *inquireOnItem*.

FIGURE 5-7 Sample system sequence diagram (SSD)

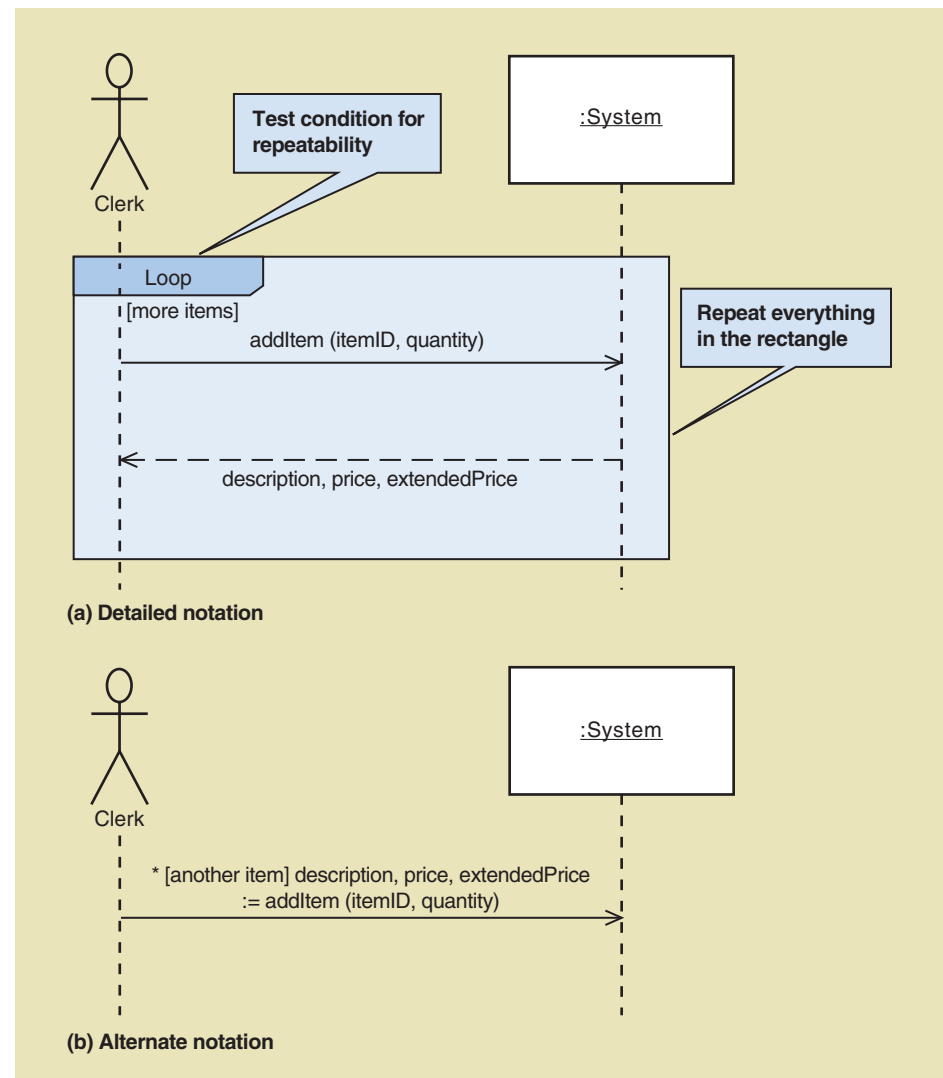


The clerk is sending a request (a message) to the system to find an item. The input data that is sent with the message is contained within the parentheses, and in this case, it is data to identify the particular item. The syntax is simply the name of the message followed by the input parameters in parentheses. This form of syntax is attached to a solid line with arrow.

The returned value has a slightly different format and meaning. Notice that the line with arrow is dashed. A dashed arrow indicates a response or an answer (in programming, a return), and as shown in the figure, it immediately follows the initiating message. The format of the label is also different. Because it is a response, only the data that are sent on the response are noted. There is no message requesting a service—only the data being returned. In this case, a valid response might be a list of all the information returned—for example, the description, price, and quantity of an item. However, an abbreviated version is also satisfactory. In this case, the information returned is named *item information*. Additional documentation is required to show the details. In Figure 5-7, this additional information is shown as a note. A note can be added to any UML diagram to add explanations. The details of item information could also be documented in supporting narratives or even simply referenced by the attributes in the Customer class.

Frequently, the same message is sent multiple times in a loop, as shown in **Figure 5-8**. For example, when an actor enters items on an order, the message

FIGURE 5-8 Repeating message in (a) detailed loop frame notation and (b) alternate notation



loop frame notation on a sequence diagram showing repeating messages

true/false condition part of a message between objects that is evaluated prior to transmission to determine whether the message can be sent

to add an item to an order may be sent multiple times. Figure 5-8(a) illustrates the notation to show this repeating operation. The message and its return are located inside a larger rectangle called a **loop frame**. In a smaller rectangle at the top of the frame is the descriptive text to control the behavior of the messages within the larger rectangle. The condition loop for all items indicates that the messages in the box repeat many times or are associated with many instances.

Figure 5-8(b) shows an alternate notation. Here, the square brackets and text inside them are called a **true/false condition** for the messages. The asterisk (*) preceding the true/false condition indicates that the message repeats as long as the true/false condition evaluates to true. Analysts use this abbreviated notation for several reasons. First, a message and the returned data can be shown in one step. Note that the return data is identified as a return value on the left side of an assignment operator—the := sign. This alternative simply shows a value that is returned. Second, the true/false condition is placed on the message itself. Note that in this example, the true/false condition is used for the control of the loop. True/false conditions are also used to evaluate any type of test that determines whether a message is sent. For example, consider the true/false condition [credit card payment]. If it is true that the thing being tested is a credit card payment, the message is sent to the system to verify a credit card number. Finally, an asterisk is placed on the message itself to indicate the message repeats. Thus, for simple repeating messages, the alternate notation is shorter. However, if several messages are included within the repeat or there are multiple messages—each with its own true/false condition—the loop frame is more explicit and precise.

Here is the complete notation for a message:

[true/false condition] return-value := message-name (parameter-list)

Any part of the message can be omitted. In brief, the notation components do the following:

- An asterisk (*) indicates repeating or looping of the message.
- Brackets [] indicate a true/false condition. This is a test for that message only. If it evaluates to true, the message is sent. If it evaluates to false, the message isn't sent.
- Message-name is the description of the requested service written as a verb-noun.
- Parameter-list (with parentheses on initiating messages and without parentheses on return messages) shows the data that are passed with the message.
- Return-value on the same line as the message (requires :=) is used to describe data being returned from the destination object to the source object in response to the message.

opt frame notation on a sequence diagram showing optional messages

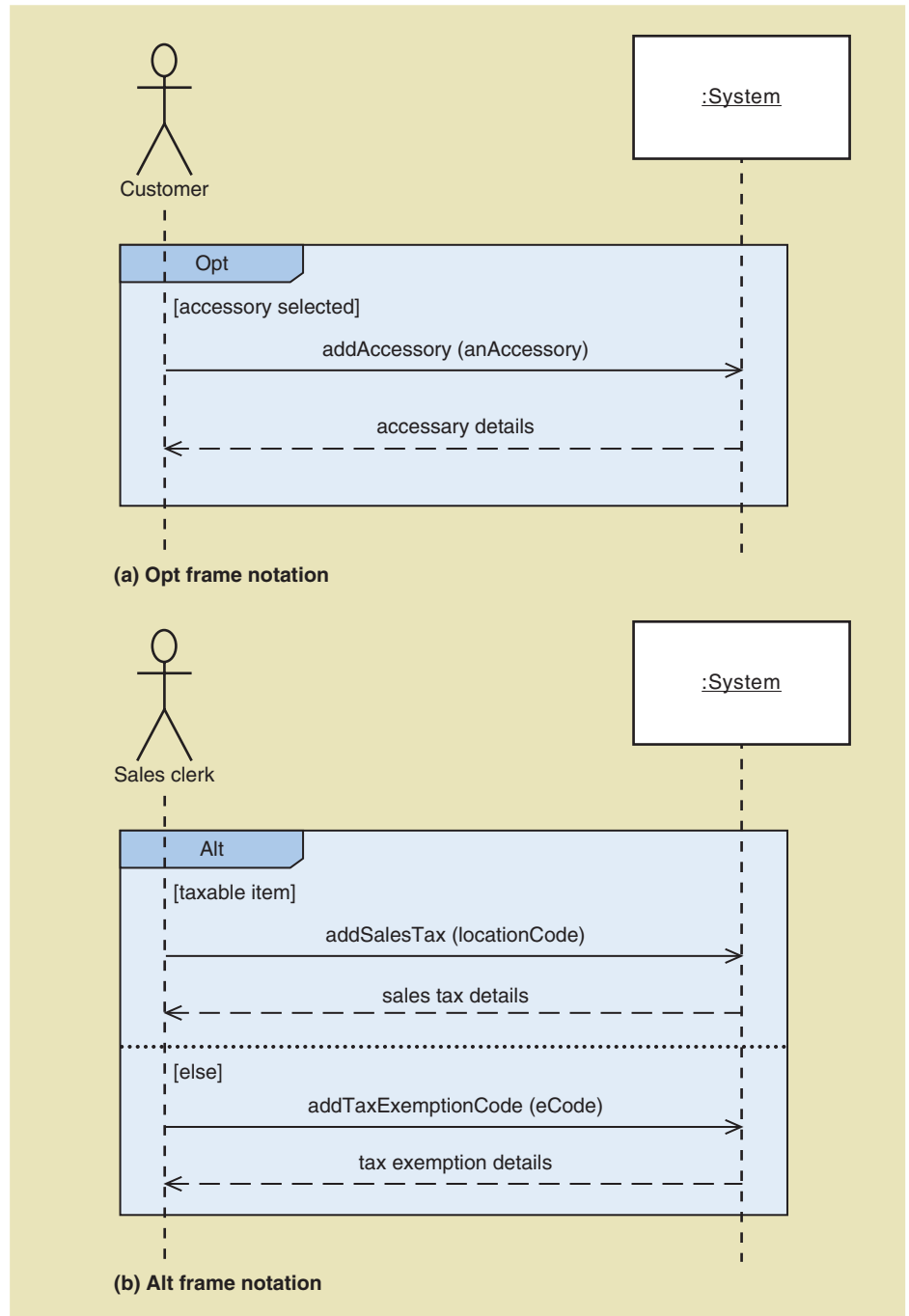
alt frame notation on a sequence diagram showing if-then-else logic

Sequence diagrams also use two addition frames to depict processing logic, as shown in **Figure 5-9**. The **opt frame** in Figure 5-9(a) is used when a message or a series of messages is optional or based on some true/false condition. The **alt frame** is used with if-then-else logic, as shown in Figure 5-9(b). The **alt frame** in this figure indicates that if an item is taxable, then add sales tax; otherwise, add a tax exemption code for a sales tax exemption.

■ Developing a System Sequence Diagram (SSD)

An SSD is usually used in conjunction with the use case descriptions to help document the details of a single use case or scenario within a use case. To develop an SSD, it is useful to have a detailed description of the use case—either in the

FIGURE 5-9 Sequence diagram notation for (a) opt frame and (b) alt frame



fully developed form or as an activity diagram. These two models identify the flow of activities within a use case, but they don't explicitly identify the inputs and outputs. An SSD will provide this explicit identification of inputs and outputs. One advantage of using activity diagrams is that it is easy to identify when an input or output occurs. Inputs and outputs occur whenever an arrow in an activity diagram goes from an external actor to the computer system.

Recall the activity diagram for *Create customer account* shown in Figure 5-4. There are two swimlanes: the customer and the computer system. In this instance, the system boundary coincides with the vertical line between the customer swimlane and the system swimlane.

The development of an SSD based on an activity diagram falls into four steps:

1. **Identify the input messages.** In Figure 5-4, there are three locations with a workflow arrow crossing the boundary line between the customer and the system. At each location that the workflow crosses the automation boundary, input data are required; therefore, a message is needed.
2. **Describe the message from the external actor to the system by using the message notation described earlier.** In most cases, you will need a message name that describes the service requested from the system and the input parameters being passed. Figure 5-10—the SSD for the *Create customer account* use case—illustrates the three messages based on the activity diagram. Notice that the names of the messages reflect the services that the actor is requesting of the system: `createNewCustomer`, `enterAddress`, and `enterCreditCard`. Other names could also have been used. For example, instead of `enterAddress`, the name could be `createAddress`. The point to remember is that the message name should describe the service requested from the system and should be in verb-noun form.

The other information required is the parameter list for each message. Determining exactly which data items must be passed in is more difficult. In fact, developers frequently find that determining the data parameters requires several iterations before a correct, complete list is obtained. The important principle for identifying data parameters is to base it on the class diagram. In other words, the appropriate attributes from the classes are listed as parameters. Looking at the attributes, along with an understanding of what the system needs to do, will help you find the right attributes. With the first message just mentioned—`createNewCustomer`—the parameters should include basic information about the customer, such as name, phone, and e-mail address. Note that when the system creates the customer, it assigns a new `customerId` and returns it with the other customer information.

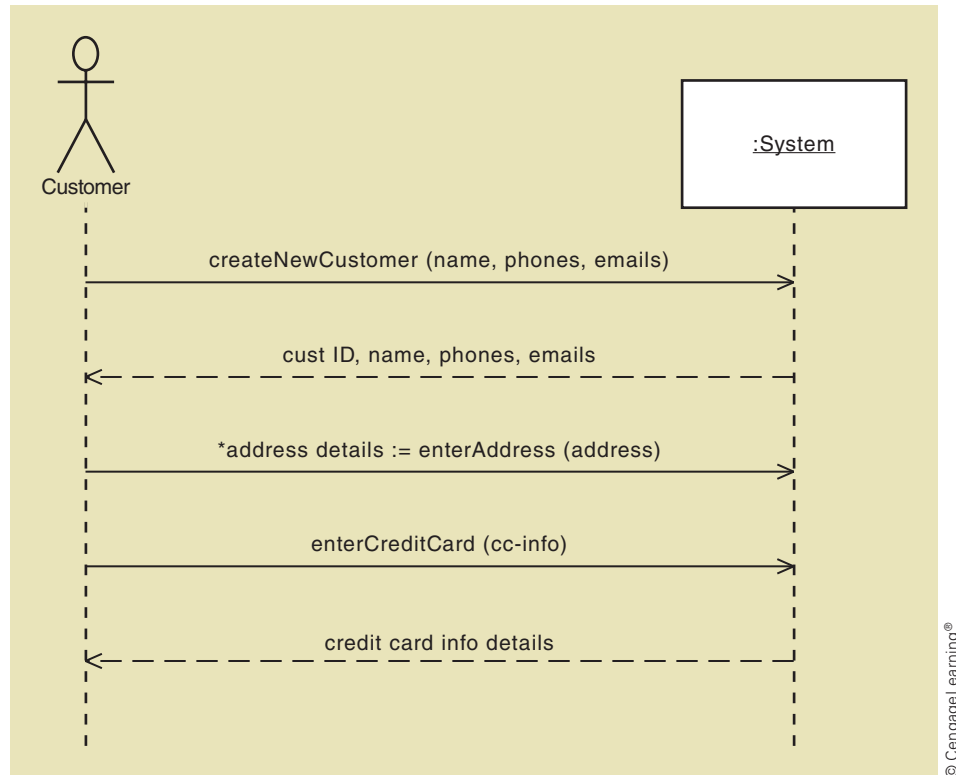
In the second message—`enterAddress`—parameters are needed to identify the full address. Usually, that would include street address, city, state, and zip code. The SSD simplifies the message to show “address” as the parameter.

The third message—based on the activity diagram—enters the credit card information. The parameter—`cc-info`—represents the account number, expiration date, and security code.

3. **Identify and add any special conditions on the input messages, including iteration and true/false conditions.** In this instance, the `enterAddress` message is repeated for each address needed for the customer. The asterisk symbol in front of the message is shown.
4. **Identify and add the output return messages.** Remember that there are two options for showing return information: as a return value on the message itself or as a separate return message with a dashed arrow. The activity diagram can provide some clues about return messages, but there is no standard rule that when a transition arrow in the workflow goes from the system to an external actor that an output always occurs. In Figure 5-4, there are three arrows from the System swimlane to the Customer swimlane. In **Figure 5-10**, these are shown as return data on the dashed line for two returns and as a returned value on the same line as the message for `enterAddress`. Note that they are each named with a noun that indicates what is being returned. Sometimes, no output data are returned.

Remember that the objective is discovery and understanding, so you should be working closely with users to define exactly how the workflow proceeds and

FIGURE 5-10 SSD for the Create customer account use case



© Cengage Learning®

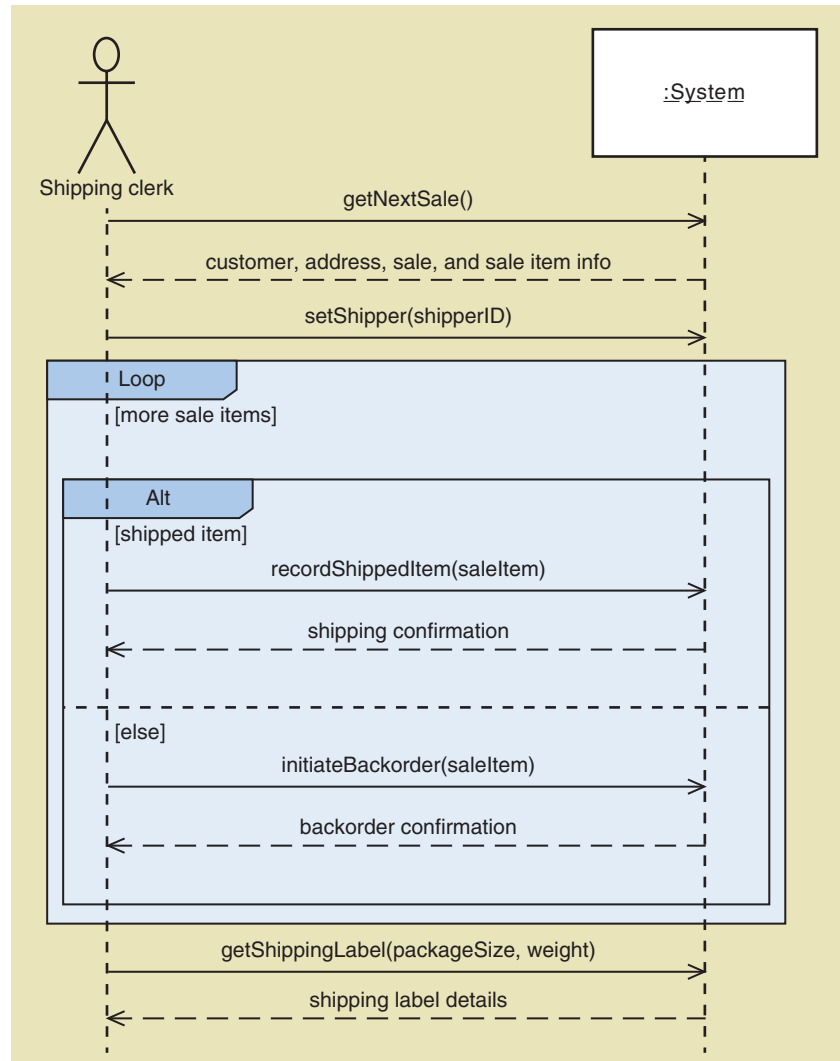
exactly what information needs to be passed in and provided as output. This is an iterative process, and you will probably need to refine these diagrams several times before they accurately reflect the needs of the users.

Let us develop an SSD for the Ship items use case that is shown as a fully developed use case description in Figure 5-3 and as an activity diagram in Figure 5-5. Note that the actor has five numbered steps in the flow of activities, so there will be five input messages in the SSD shown in **Figure 5-11**: getNextSale, setShipper, recordShippedItem, initiateBackorder, and getShippingLabel. No parameter is needed for getNextSale because the system will automatically return the information for the next sale to be shipped. The shipper is selected by the actor—probably from a list on the form or page—so the parameter is shipperID.

There is a loop frame that repeats for each sale item in the sale. Inside the loop frame is an alt frame that tests the whether each sale item is available to ship. If the item can be shipped, the recordShippedItem message is sent to the system. If the item cannot be shipped because it is out of stock or perhaps damaged, the initiateBackorder message is sent to the system. Finally, the getShippingLabel message requires two parameters: the size of the package and the weight. The system uses that information, along with the shipper and address, to produce the shipping label and record the cost.

These first sections of this chapter have explained the models that are used in object-oriented development to specify the processing aspects of the new system. The use case descriptions, as provided by written narratives or activity diagrams, give the details of the internal steps within each use case. Precondition and postcondition statements help define the context for the use case—that is, what must exist before and after processing. Finally, the SSD describes the inputs and outputs that occur within a use case. Together, these models provide a comprehensive description of the system-processing requirements and give the foundation for systems design.

FIGURE 5-11 SSD for the Ship items use case



■ Use Cases and CRUD

CRUD technique an acronym for Create, Read/Report, Update, and Delete; a technique to validate or refine use cases

Another important technique used to validate use cases is the **CRUD technique**, which involves verifying that all of the needed use cases have been identified to maintain the data represented by the domain model class diagram. **CRUD** is an acronym for Create, Read or Report, Update, and Delete, and it is often introduced with respect to database management. The analyst starts by looking at the types of data stored by the system, which are modeled as domain classes, as described in Chapter 4. In the RMO Tradeshow System discussed in Chapter 1, the types of data included Supplier, Contact, Product, and ProductPicture. In the RMO CSMS, the types of data include Customer, Sale, Inventory Item, Promotion, Shipment, and many others. To validate and refine use cases, the analyst looks at each type of data and verifies that use cases have been identified that create the data, read or report on the data, update the data, and delete (or archive) the data.

The CRUD technique is most useful when used as a cross-check along with the user goal technique. Users will focus on their primary goals, and use cases that update or archive data will often be overlooked. The CRUD technique makes sure all possibilities are identified. Sometimes, domain classes are shared by a set of integrated applications. For example, RMO has a supply chain management application that is responsible for managing inventory levels and adding products. The RMO CSMS will not need to create or delete products, but it will need to look up and update product information. It is important to identify

FIGURE 5-12 Verifying use cases with the CRUD technique

Data entity/domain class	CRUD	Verified use case
Customer	Create	Create customer account
	Read/report	Look up customer Produce customer usage report
	Update	Process account adjustment Update customer account
	Delete	Update customer account (to archive)

© CengageLearning®

the other application that is responsible for creating, updating, or deleting the data to be clear about the scope of each system. **Figure 5-12** shows an example of potential use cases based on the CRUD technique for RMO Customer data.

Note in **Figure 5-12** that the analyst has not blindly added a use case to create, read/report, update, and delete instances of a customer. The CRUD technique is best used to take already identified use cases and verify that there are use cases for create, read, update, and delete as a cross-check. *Create customer account* creates customer objects. A separate *Create customer* use case is not needed. *Update customer account* is defined to archive a customer, so no separate *Archive customer use case* is required.

Another use of the CRUD technique is to summarize all use cases and all data entities/domain classes to show the connection between use cases and data. In **Figure 5-13**, some of the use cases are matched with domain classes by including *C*, *R*, *U*, or *D* in the cell corresponding to the role of the use case in terms of data. For example, the use case *Create customer account* actually creates customer data and account data, so the *C* is included in those two cells. The use case *Process account adjustment* reads information about the sale, reads information about the customer, updates the account, and creates an adjustment.

Based on the information shown in **Figure 5-13**, there are insufficient use cases to cover the Sale and the Adjustment domain classes. The Sale class will need to have additional use cases to create, update, and delete Sale objects. In addition, the Adjustment class will require use cases to update, report, and delete Adjustment objects.

The CRUD technique for validating and refining use cases includes these steps:

1. Identify all the domain classes involved in the new system. Chapter 4 discussed these in more detail.
2. For each type of data (domain class), verify that a use case has been identified that creates a new instance, updates existing instances, reads or reports values of instances, and deletes (or archives) an instance.
3. If a needed use case has been overlooked, add a new use case and then identify the stakeholders.
4. With integrated applications, make sure it is clear which application is responsible for adding and maintaining the data and which system merely uses the data.

FIGURE 5-13 CRUD table showing use cases and corresponding domain classes

Use case vs. entity/domain class	Customer	Account	Sale	Adjustment
Create customer account	C	C		
Look up customer	R	R		
Produce customer usage report	R	R	R	
Process account adjustment	R	U	R	C
Update customer account	UD (archive)	UD (archive)		

© CengageLearning®

■ Integrating Requirements Models

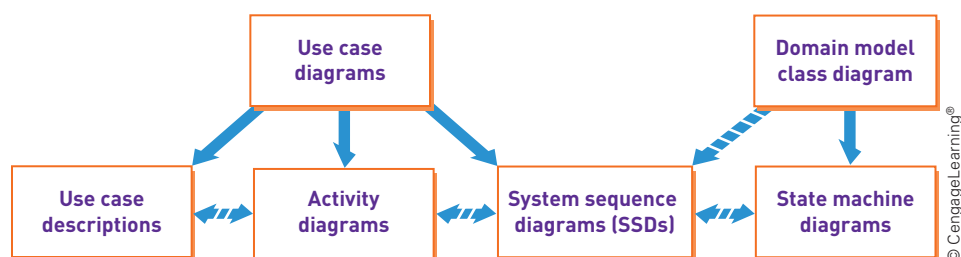
The diagrams described in this chapter allow analysts to completely specify the system functional requirements by modeling details of the use cases. Because you are using an iterative approach, you would only construct the diagrams that are necessary for a given iteration. A complete use case diagram would be important to get an idea of the total scope of the new system. But the supporting details included in use case descriptions, activity diagrams, and system sequence diagrams need only be done for use cases in the specific iteration. Again, it is important to remember that not all use cases need to be modeled in detail.

The domain model class diagram is a special case. Much like the entire use case diagram, the domain model class diagram should be as complete as possible for the entire system early in the project, as shown for RMO in Chapter 4. The number of problem domain classes for the system provides an additional indicator of the total scope of the system. Refinement and actual implementation of many classes will wait for later iterations, but the domain model should be fairly complete. The domain model is necessary to identify all the domain classes that are required in the new system. The domain model is also used to design the database.

Throughout this chapter, you have seen how the construction of a diagram depends on information provided by another diagram. You have also seen that the development of a new diagram often helps refine and correct a previous diagram. You should also have noted that the development of detailed diagrams is critical to gaining a thorough understanding of the user requirements. **Figure 5-14** illustrates the primary relationships among the requirements models for object-oriented development. The use case diagram and other diagrams on the left are used to capture the processes of the new system. The class diagram and its dependent diagrams capture information about the classes for the new system. The solid arrows represent major dependencies, and the dashed arrows show minor dependencies. The dependencies generally flow from top to bottom, but some arrows have two heads to illustrate that influence goes in both directions.

Note that the use case diagram and the domain model class diagram are the primary models from which others draw information. You should develop those two diagrams as completely as possible. The detailed descriptions—either in narrative format or in activity diagrams—provide important internal documentation of the use cases and must completely support the use case diagram. Such internal descriptions as preconditions and postconditions use information from the domain model class diagram. These detailed descriptions are also important for development of system sequence diagrams. Thus, the detailed descriptions, activity diagrams, and system sequence diagrams must all be consistent with regard to the steps of a particular use case. As you progress in developing the system and especially as you begin doing detailed systems design, you will find that understanding the relationships among these models is an important element in the quality of your models.

FIGURE 5-14 Relationships among object-oriented requirements models



CHAPTER SUMMARY

The object-oriented approach to information systems development has a complete set of diagrams and textual models that together document the user's needs and define the system requirements. These requirements are specified by using domain model class diagrams and state machine diagrams to model the domain classes and use case diagrams, use case descriptions or activity diagrams, and system sequence diagrams (SSDs) to model the use cases.

The internal activities of a use case are first described by an internal flow of activities. It is possible to have several different internal flows, which represent different scenarios of the same use case. Thus, a use case may have several scenarios. These details are documented either in use case descriptions or with activity diagrams.

Another diagram that provides details of the use case's processing requirements is an SSD. An SSD documents the inputs and outputs of the system.

The scope of each SSD is usually a use case or a scenario within a use case. The components of an SSD are the actor—the same actor identified in the use case—and the system. The system is treated as a black box in that the internal processing isn't addressed. Messages, which represent the inputs, are sent from the actor to the system. Output messages are returns from the system to the actor. The sequence of messages is indicated from top to bottom.

The CRUD technique is used to validate that all of the use cases have been identified that are necessary to maintain the data represented by the domain model class diagram. Use cases must be identified that create, read or report, update, and delete or archive instances of all classes. Alternatively, other systems that maintain the data must be identified when integrated systems share the same database. All the models discussed in this chapter are interrelated, and information in one model explains information in others.

KEY TERMS

alt frame

CRUD technique

lifeline or object lifeline

loop frame

opt frame

postconditions

preconditions

scenarios or use case instances

system sequence diagram (SSD)

true/false condition

use case description

REVIEW QUESTIONS

1. What are the models that describe use cases in more detail?
2. What two UML diagrams are used to model domain classes?
3. Which part of a use case description can also be modeled by using an activity diagram?
4. Explain the difference between a use case and a scenario. Give a specific example of a use case with a few possible scenarios.
5. List the parts or compartments of a fully developed use case description.
6. Compare/contrast precondition and postcondition.
7. Compare/contrast postcondition and exception condition.
8. Compare/contrast business process and flow of activities for a use case. Explain how an activity diagram can be used to model both.
9. What is the purpose of an SSD? What symbols are used in an SSD?
10. What are the steps required to develop an SSD?
11. Write a complete SSD message from the actor to the system, with the actor asking the system to begin the process for updating information about a specific product.
12. What is the name of the sequence diagram symbol used to represent the object's existence throughout the duration of a use case?
13. What are the two ways to show a returned value on a sequence diagram?

14. What are two ways to show repetition on a sequence diagram?
15. What are the three types of frames used on a sequence diagram?
16. What is the symbol for a true/false condition on a sequence diagram?
17. Explain what parameters of a message are.
18. List the primary steps for developing an SSD.
19. What are the words included in the CRUD acronym?
20. What is the purpose of using the CRUD technique?
21. Identify the models explained in this chapter and their relationship to one another.

PROBLEMS AND EXERCISES

1. After reading the following narrative, do the following:

- a. Develop an activity diagram for each scenario.
- b. Complete a fully developed use case description for each scenario.

Quality Building Supply has two kinds of customers: contractors and the general public. Sales to each are slightly different.

A contractor buys materials by taking them to the checkout desk for contractors. The clerk enters the contractor's name into the system. The system displays the contractor's information, including current credit standing. The clerk then opens up a new ticket (sale) for the contractor. Next, the clerk scans in each item to be purchased. The system finds the price of the item and adds the item to the ticket. At the end of the purchase, the clerk indicates the end of the sale. The system compares the total amount against the contractor's current credit limit and, if it is acceptable, finalizes the sale. The system creates an electronic ticket for the items, and the contractor's credit limit is reduced by the amount of the sale. Some contractors like to keep a record of their purchases, so they request that ticket details be printed. Others aren't interested in a printout.

A sale to the general public is simply entered into the cash register, and a paper ticket is printed as the items are identified. Payment can be made by cash, check, or credit card. The clerk must enter the type of payment to ensure that the cash register balances at the end of the shift. For credit card payments, the system prints a credit card voucher that the customer must sign.

2. Based on the following narrative, develop either an activity diagram or a fully developed description for the use case of *Add a new vehicle to an existing policy* in a car insurance system.

A customer calls a clerk at the insurance company and gives his policy number. The clerk enters this information, and the system displays the basic insurance policy. The clerk then checks

the information to make sure the premiums are current and the policy is in force.

The customer gives the make, model, year, and vehicle identification number (VIN) of the car to be added. The clerk enters this information, and the system ensures that the given data are valid. Next, the customer selects the types of coverage desired and the amount of each. The clerk enters the information, and the system records it and validates the requested amount against the policy limits. After all the coverages have been entered, the system ensures the total coverage against all other ranges, including other cars on the policy. Finally, the customer must identify all the drivers and the percentage of time they drive the car. If a new driver is to be added, then another use case—*Add new driver*—is invoked.

At the end of the process, the system updates the policy, calculates a new premium amount, and prints the updated policy statement to be mailed to the policy owner.

3. Given the following list of classes and associations for the previous car insurance system, list the preconditions and postconditions for the use case *Add a new vehicle* to an existing policy.

Classes in the system include:

- Policy
- InsuredPerson
- InsuredVehicle
- Coverage
- StandardCoverage (lists standard insurance coverages with prices by rating category)
- StandardVehicle (lists all types of vehicles ever made)

Relationships in the system include:

- Policy has InsuredPersons (one-to-many)
- Policy has InsuredVehicles (one-to-many)
- Vehicle has Coverages (one-to-many)
- Coverage is a type of StandardCoverage
- Vehicle is a StandardVehicle

4. Develop an SSD based on the narrative and your activity diagram for problem 1.
5. Develop an SSD based on the narrative or your activity diagram for problem 2.
6. Locate a company in your area that develops software. Consulting companies or companies with a large staff of information systems professionals tend to be more rigorous in their approach to system development. Set up an

interview. Determine the development approaches that the company uses. Many companies still use traditional structured techniques combined with some object-oriented development. In other companies, some projects are structured, whereas other projects are object oriented. Find out what kinds of modeling the company does for requirements specification. Compare your findings with the techniques taught in this chapter.

CASE STUDY

TheEyesHavelt.com Book Exchange

TheEyesHavelt.com Book Exchange is a type of e-business exchange that does business entirely on the Internet. The company acts as a clearinghouse for buyers and sellers of used books.

To offer books for sale, a person must register with TheEyesHavelt.com. The person must provide a current physical address and telephone number as well as a current e-mail address. The system then maintains an open account for this person. Access to the system as a seller is through a secure, authenticated portal.

A seller can list books on the system through a special Internet form. The form asks for all the pertinent information about the book: its category, its general condition, and the asking price. A seller may list as many books as desired. The system maintains an index of all books in the system so buyers can use the search engine to search for books. The search engine allows searches by title, author, category, and keyword.

People who want to buy books come to the site and search for the books they want. When they decide to buy, they must open an account with a credit card to pay for the books. The system maintains all this information on secure servers.

When a purchase is made, TheEyesHavelt.com sends an e-mail notice to the seller of the book that was chosen as well as payment information. It also marks the book as sold. The system maintains an open order until it receives

notice that the book has been shipped. After the seller receives notice that a listed book has been sold, the seller must notify the buyer via e-mail within 48 hours that the purchase is noted. Shipment of the order must be made within 24 hours after the seller sends the notification e-mail. The seller sends a notification to the buyer and TheEyesHavelt.com when the shipment is made.

After receiving the notice of shipment, TheEyesHavelt.com maintains the order in a shipped status. At the end of each month, a check is mailed to each seller for the book orders that have remained in a shipped status for 30 days. The 30-day waiting period exists to allow the buyer to notify TheEyesHavelt.com if the shipment doesn't arrive for some reason or if the book isn't in the same condition as advertised.

If they want, buyers can enter a service code for the seller. The service code is an indication of how well the seller is servicing book purchases. Some sellers are very active and use TheEyesHavelt.com as a major outlet for selling books. Thus, a service code is an important indicator to potential buyers.

For this case, develop these diagrams:

1. A domain model class diagram
2. A list of uses cases and a use case diagram
3. A fully developed description for two use cases: *Add a seller* and *Record a book order*
4. An SSD for each of the two use cases: *Add a seller* and *Record a book order*

RUNNING CASE STUDIES

Community Board of Realtors®

The Multiple Listing Service system has a number of use cases, which you identified in Chapter 3, and three key domain classes, which you identified in Chapter 4: RealEstateOffice, Agent, and Listing.

1. For the use case *Add agent to real estate office*, write a fully developed use case description. Also develop an activity diagram and draw an SSD. Review the case materials in previous

chapters and recall that the system will need to know which real estate office the agent works for before prompting for agent information.

2. For the use case *Create new listing*, write a fully developed use case description. Also develop an activity diagram and draw an SSD. Recall that

the system needs to know which agent made the listing before the system prompts for listing information.

3. Do a CRUD analysis based on the use cases and classes you have identified. What does it show?

The Spring Breaks ‘R’ Us Travel Service

The Spring Breaks ‘R’ Us Travel Service system has many use cases and domain classes, which you identified in Chapters 3 and 4.

1. For the use case *Add new resort*, write a fully developed use case description and draw an SSD. Review the classes that are associated with a resort in the domain model to understand the flow of activities and repetition involved.
2. For the use case *Book a reservation*, write a fully developed use case description and draw an SSD.

Review the classes that are associated with a reservation in the domain model to understand the flow of activities and repetition involved.

3. Draw an activity diagram to show the flow of activities for the use case *Add a new resort*.
4. Do a CRUD analysis based on the domain model you build in Chapter 4 and the use cases you have defined thus far.

On the Spot Courier Services

As On the Spot Courier Services continues to grow, Bill discovers that he can provide much better services to his customers if he utilizes some of the technology that is currently available. For example, it will allow him to maintain frequent communication with his delivery trucks, which could save transportation and labor costs by making the pickup and delivery operations more efficient. This would allow him to serve his customers better. Of course, a more sophisticated system will be needed, but Bill’s development consultant has assured him that a straightforward and not-too-complex solution can be developed.

Here is how Bill wants his business to operate. Each truck will have a morning and afternoon delivery and pickup run. Each driver will have a portable digital device with a touch screen. The driver will be able to view his or her scheduled pickups and deliveries for that run. (Note: This process will require a new use case—something the Agile development methodology predicted would happen.) However, because the trucks will maintain frequent contact with the home office via telephony Internet access, the pickup/delivery schedule can be updated in real time—even during a run. Rather than maintain constant contact, Bill decides that it will be sufficient if the digital device synchronizes with the home office whenever a pickup or delivery is made. At those points in time, the route schedule can be updated with appropriate information.

Previously, customers were able to either call On the Spot and request a package pickup or visit the company’s Web site to schedule a pickup. Once customers

logged in, they could go to a Web page that allowed them to enter information about each package, including “deliver to” addresses, size and weight category information, and type of service requested. On the Spot provided “three hour,” “same day,” and “overnight” services. To facilitate customer self-service, On the Spot didn’t require exact weights and sizes, but there were predefined size and weight categories from which the customer could choose.

Once the customer entered the information for all the packages, the system would calculate the cost and then print mailing labels and receipts. Depending on the type of service requested and the proximity of a delivery truck, the system would schedule an immediate pickup or one for later that day. It would display this information so the customer would immediately know when to expect the pickup.

Picking up packages was a fairly straightforward process. But there was some variation in what would happen depending on what information was in the system and whether the packages were already labeled. Upon arriving at the scheduled pickup location, the driver would have the system display any package information available for this customer. If the system already had information on the packages, the driver would simply verify that the correct information was already in the system for the packages. The driver could also make such changes as correcting the address, deleting packages, or adding new packages. If this were a cash customer, the driver would collect any money and enter that into the system. Using a portable

printer from the van, the driver could print a receipt for the customer as necessary. If there were new packages that weren't in the system, the driver would enter the required information and also print mailing labels with his portable printer.

One other service that customers required was to be able to track the delivery status of their packages. The system needed to track the status of a package from the first time it “knew” about the package until it was delivered. Such statuses as “ready for pickup,” “picked up,” “arrived at warehouse,” “out for delivery,” and “delivered” were important. Usually, a package would follow through all the statuses, but due to the sophistication of the scheduling and delivery algorithm, a package would sometimes be picked up and delivered on the same delivery run. Bill also decided to add a status of “canceled” for those packages that

were scheduled to be picked up but ended up not being sent.

1. Based on this description, develop the following for the use case *Request a package pickup* and for the Web customer scenario:
 - a. A fully developed use case description
 - b. An activity diagram
 - c. An SSD

Based on the same description, develop the following for the use case *Pickup a package*:

- a. A fully developed use case description
 - b. An activity diagram
 - c. System sequence diagram
2. Based on the domain model and the list of use cases you developed in Chapters 3 and 4, do a CRUD analysis for each of the identified classes.

Sandia Medical Devices Real-Time Glucose Monitoring

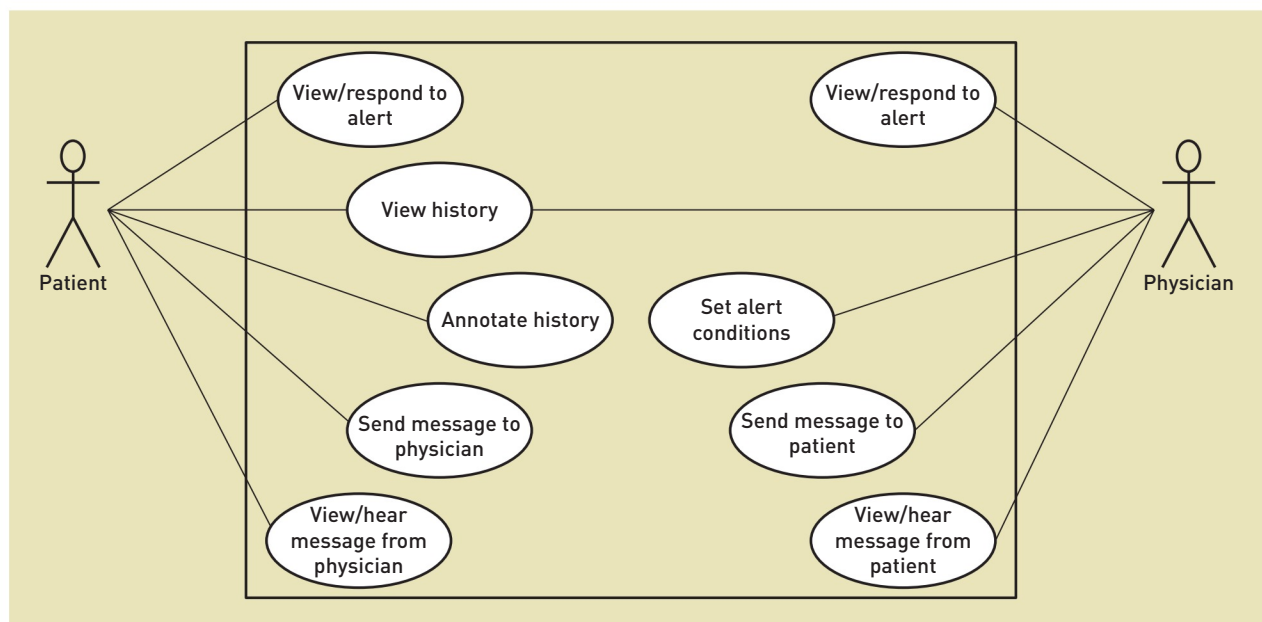
Figure 5-15 shows a set of use cases for the patient and physician actors. Answer the following questions and/or complete the following exercises:

1. Which use cases include which other use cases? Modify the diagram to incorporate included relationships.
2. Consider the use cases *View/respond to alert* and *View history*. Both actors share the latter, but each has a different version of the former. Why do the actors have different versions of the *View/*

respond to alert use case? Would the diagram be incorrect if each actor had his own version of the *View history* use case? Why or why not?

3. Develop an SSD for the *View history* use case. Assume that the system will automatically display the most recent glucose level, which is updated at five-minute intervals by default. Assume further that the user can ask the system to view glucose levels during a user-specified time period and that the levels can be displayed in tabular form or as a graph.

FIGURE 5-15 RTGM system use case diagram



FURTHER RESOURCES

Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

E. Reed Doke, J. W. Satzinger, and S. R. Williams, *Object-Oriented Application Development Using Java*. Course Technology, 2002.

Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado, *UML 2 Toolkit*. John Wiley & Sons, 2004.

Martin Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (3rd ed.). Addison-Wesley, 2004.

Philippe Kruchten, *The Rational Unified Process: An Introduction* (3rd ed.). Addison-Wesley, 2005.

Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process* (3rd ed.). Prentice Hall, 2005.

Object Management Group, *UML 2.0 Superstructure Specification*. 2004.



Essentials of Systems Design

PART THREE

- **Chapter 6**
Foundations for Systems Design
- **Chapter 7**
Defining the System Architecture
- **Chapter 8**
Designing the User Interface
- **Chapter 9**
Designing the Database

Foundations for Systems Design

CHAPTER SIX

CHAPTER OUTLINE

- What Is Systems Design?
- Design Activities
- System Controls and Security

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- Describe systems design and contrast it with systems analysis
- List documents and models used as inputs to or output from systems design
- Explain each major design activity
- Describe security methods and controls

OPENING CASE SECURITY AND CONTROLS AT NEW MEXICO HEALTH SYSTEMS

Jim Gutierrez was hired three months ago by New Mexico Health Systems (NMHS) to oversee all software development projects. Though Jim lacked prior experience in health-care services, he'd been hired due to excellent performance in software development projects for a variety of firms in various industries. His skills in designing and deploying complex mobile applications were especially valuable for several projects in progress at NMHS, including a reimplementing of the billing and insurance reimbursement system and an upgrade to the patient portal that enables patients to view their medical records and to interact with their primary care providers via a secure messaging system.

From his first week on the job, Jim was concerned about security and controls within NMHS's existing systems and those under development. Jim gained little traction with senior management when he raised those concerns, but they did agree to let him hire an outside consultant. Consequently, he'd hired Alice Watts, a security consultant with extensive health-care-related experience, to complete a thorough analysis of existing systems and to make recommendations for securing the systems under development. Alice's analysis had taken almost a month to complete. Jim had received some disturbing preliminary findings as the work progressed, but he hadn't yet seen the complete report.

At a meeting that included upper management, Alice presented her report. She began, "Though I'll cover several security-related issues in my presentation, I want to start by saying that there is far more good news than bad about the security of your infrastructure and your existing systems. You have an excellent network administrator. She's hardened your network and the systems within it against malicious attacks. Bright spots include extensive encryption of data at rest and in transit, robust user authentication and authorization methods, and a well-implemented system of access controls for your hardware, operating systems, databases, and applications."

Alice continued, "Though your infrastructure security methods do much to protect your systems from outside attack, I did find that internal controls within specific systems are a hit-or-miss proposition. For example, your system for transcribing recorded notes by physicians and other service providers doesn't incorporate the latest dictionary-based technology to detect data-entry errors. Also, there's no requirement for review or approval by a supervisor or the health-care provider. Because transcription is an error-prone process, you

need additional controls to ensure the integrity of patient medical records—especially for follow-up orders for lab tests, prescriptions, and downstream care coordination."

Rajesh Kumar (the chief financial officer) interrupted, "We're planning a pilot for an expensive text-to-speech system that will eliminate the need for transcription. We were hoping to avoid spending any more time and resources on the current system."

Alice replied, "While I understand the sentiment, you're opening yourself to medical errors and lawsuits in the interim. One transcription-related malpractice suit could easily cost you far more than technology updates for the current system."

Jim added, "I've investigated this issue further since Alice alerted me a couple of weeks ago. I think that we can make the needed changes in a matter of weeks for about \$20,000."

Diana Lourdes (the chief medical officer) replied while glancing at Rajesh, "I think that's a prudent investment."

Alice continued, "There are a few other easy-to-resolve issues with existing systems. But I'd like to skip over them to what I think are much more important topics. My biggest security-related concerns are with your two systems under development—especially the patient portal. Both systems open internal data to outside access. The billing application has some weaknesses despite controls implemented within the Web services interface. But the patient portal is your biggest risk because it breaks new ground by enabling mobile browser-based access from ordinary users with untrusted devices. Your current design, development, and deployment methods aren't up to the task of fully securing such a system. The risk of malicious attack and accidental data release through the patient portal is very high. You can't fully mitigate that risk with infrastructure-related security and controls bolted on to the system after it's been developed. You have to design the security and controls into every part of the software and then layer additional security on top."

Rajesh interrupted with considerable concern, "Is this project in danger? Do we need to start over?"

Alice replied, "I wouldn't go that far, but some immediate changes are needed. I'll provide an executive summary of my recommendations here. Then, I'll review my detailed findings with Jim over lunch and with his development staff this afternoon. As part of that discussion, we'll revisit some of the design decisions made thus far, determine the impact on work already completed, and reassess the target deployment date."

■ Overview

Previous chapters described the activities and decisions associated with discovering and understanding the major elements of the user's requirements—in other words, the analysis activities. This chapter focuses on the solution system. During analysis, the focus is on *understanding* what the system should do (i.e., the requirements), whereas during design, the focus is on the *solution* (i.e., specifying how the system will be built and what the structural components of the new system will be).

New developers often ask, “When are these tasks carried out in a real project?” Unfortunately, there is no single answer. Many projects begin with some of the design decisions having already been made, particularly with regard to the deployment environment, when companies already have a strong technology infrastructure in place. For other projects, the new system may be the result of a new thrust for the organization, and thus the decisions are wide open. However, it is normal for the project team to start thinking about these issues very early in development and to begin making preliminary decisions as requirements are being defined. The topics discussed in this and the following chapters are solution-oriented design topics; however, you shouldn't try to come up with a complete solution until you understand the problem.

This is the first of several chapters that discuss design. This chapter briefly describes all the design activities and discusses the overarching issues of security and controls in more detail. Later chapters explore other design activities and explain in detail the various models and techniques used for systems design.

■ What Is Systems Design?

The term *design* has different meanings depending on the context in which it is used. For example, when designing a poster or banner, the designer is primarily concerned with the message, text, and images, including their placement and stylistic elements, such as fonts and colors. The designer may also consider materials, such as paper for a small poster or tough and resilient plastics for a large banner stretched across a street. A good poster or banner design captures the viewer's attention, clearly communicates the intended message in a way that leaves a lasting impression, and produces a final product durable enough to last as long as needed. The entire design can typically be completed by one person in a matter of hours or days.

In contrast, design of a commercial aircraft is much more complex and concerned with practical and functional aspects more than with stylistic ones. Design typically starts with macro-level considerations, such as the aircraft's shape, size, and the number and placement of engines. From those decisions, designers move on to lower-level details and subsystems, such as internal engine mechanics, control surfaces, the fuel storage/delivery system, hydraulic flight controls, and myriad electrical subsystems. Designers also focus on minute details, such as the placement of rivets and welds, the size and shape of seats, and the placement of cockpit display and control devices. Because aircraft are so complex, multiple designers work over a period of months or years and coordinate their efforts with extensive drawings, mathematical models, and other design documents. The drawings and models also provide a way for designers to communicate with project managers, parts suppliers, designers of production machinery, and the crews that will eventually build the planes.

■ Analysis, Design, and Implementation

Chapter 1 defined the term *systems design* as “those system development activities that enable a person to describe in detail how the resulting information system will actually be implemented.” That definition was preceded by a definition of *systems analysis*: “those system development activities that enable a person to understand and specify what the new system should accomplish.” The sequence of the definitions implies that analysis precedes design. That is, to design anything, you must first understand and specify what the final product will accomplish. In the case of a banner or poster, analysis describes the message to be conveyed and physical aspects, such as physical size and where it will be displayed. In the case of the commercial aircraft, analysis determines requirements, such as carrying capacity, fuel consumption, range, maximum speed, reliability, maintenance costs, longevity, and maximum construction cost. Those complex and interdependent requirements drive a complex and interdependent set of design tasks.

The latter part of the systems design definition, “describe...how the resulting information system will actually be implemented,” implies a direct and sequential link between design and implementation. That is, designers provide the plans that others follow to build the final product or system. In the case of a banner or poster, the designer provides a full-size or reduced-scale drawing and a set of construction specifications, such as size, material, thickness, and placement of mounting grommets. The entire design is described in a few pages. In contrast, the commercial aircraft design is described in thousands of documents and drawings that include detailed specifications for every part. In both cases, design results are recorded in documents and models that provide sufficient information to build the final product.

Figure 6-1 summarizes aspects of design that you can glean from the poster/banner and aircraft examples and apply to information systems design:

- Analysis provides the starting point for design.
- Design provides the starting point for implementation.
- Analysis and design document their results to coordinate the work of multiple people and to communicate with downstream activities.

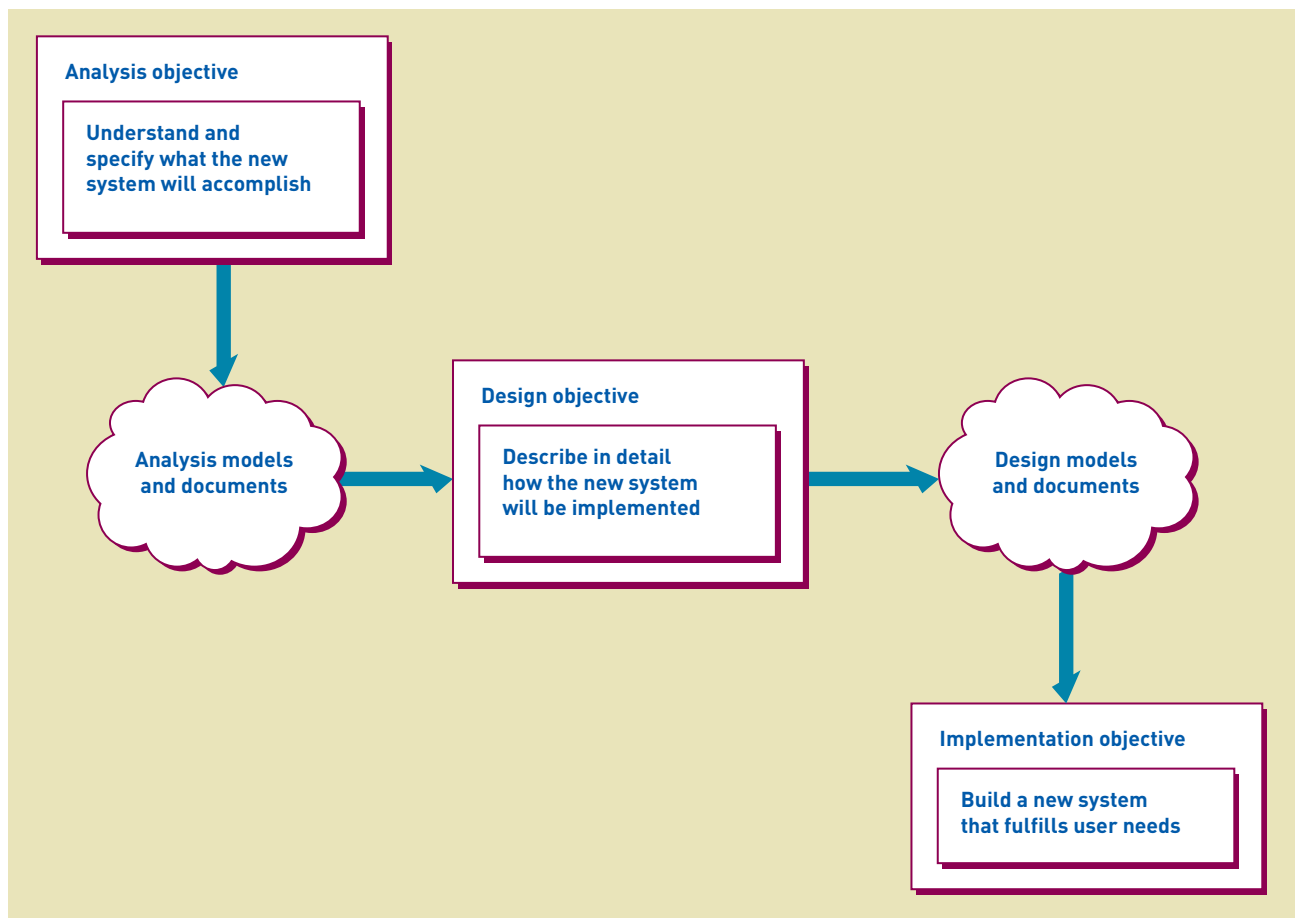
■ Design Models

During the analysis activities described in previous chapters, we created documents and models. For object-oriented analysis, we create models such as class diagrams, use case diagrams, use case descriptions, activity diagrams, system sequence diagrams, and state machine diagrams. The Online Chapter B, “The Traditional Approach to Requirements,” presents the traditional analysis models, such as the event table, data flow diagrams, and entity-relationship diagrams. Regardless of the approach, design activities begin with documents and models created during earlier analysis activities. In other words, analysis outputs are design inputs.

In iterative projects, analysis and design activities are often done concurrently. However, the first focus of any iteration has to be identifying and specifying the requirements for some part of the system (i.e., analysis); determining the solutions (i.e., design) comes later.

During analysis, analysts build models to represent the real world and to understand business processes and the information used in those processes. Basically, analysis involves decomposition—breaking a complex problem with complicated information requirements into smaller, more understandable pieces. Analysts then organize, structure, and document the problem domain knowledge by building requirements models. Analysis and modeling require substantial user involvement to explain the requirements and to verify that the models are accurate.

FIGURE 6-1 Models and documents link analysis, design, and implementation

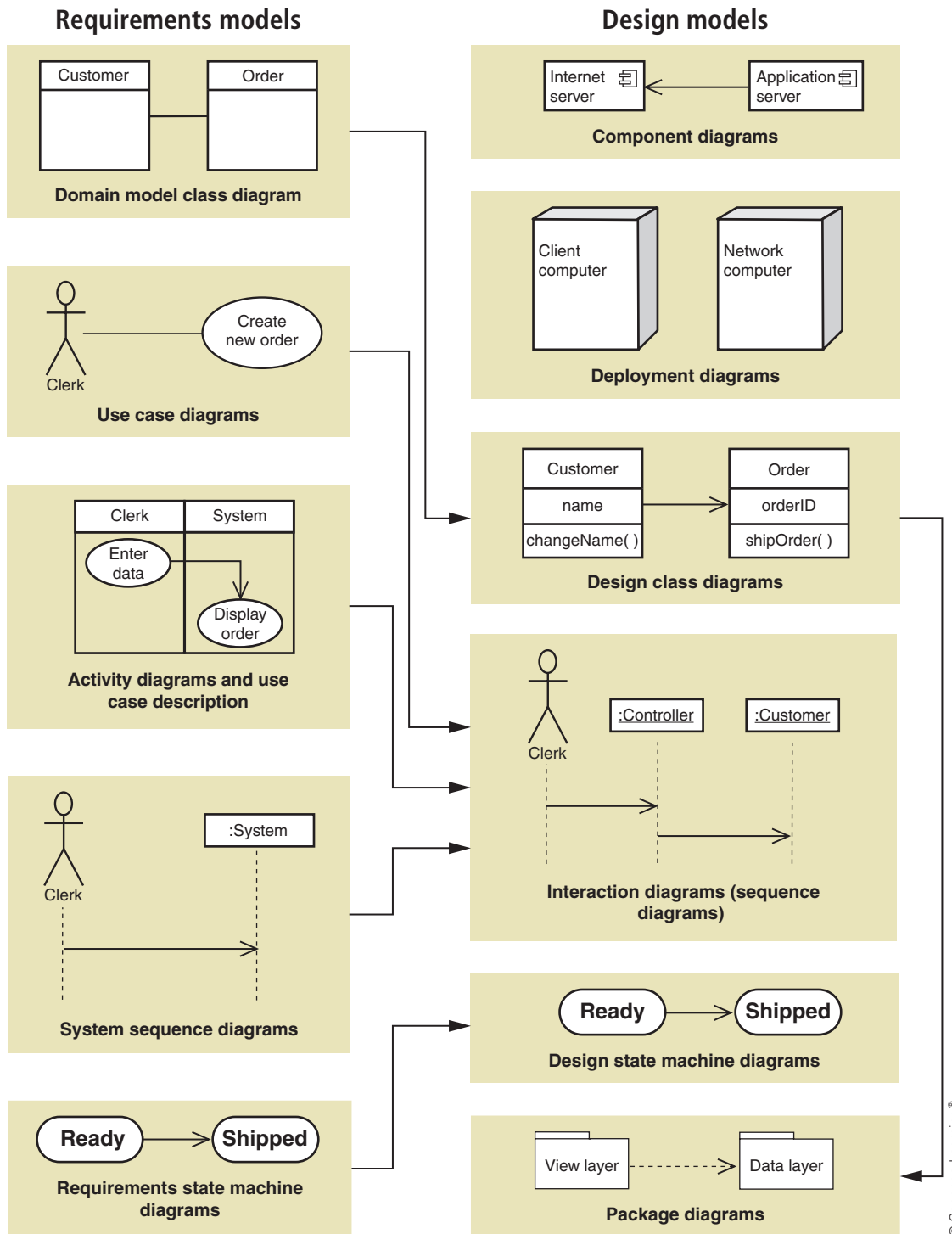


Design is also a model-building activity. Designers convert the information gathered during analysis—the requirements models—into models that represent the solution system. Design is much more oriented toward technical issues, and therefore requires less user involvement and more involvement by other systems professionals. The output of design activities is a set of diagrams and documents that models various aspects of the solution system.

The formality of a project also affects design. Formal projects usually require well-developed design documents typically reviewed in formal meetings. Developers on informal projects often create their designs with notepads and pencils and then throw away the design once the program is coded. This kind of informal design (used in many Agile projects) is merely a means to an end, which is the actual program code. However, even in Agile development, the design activities described in this chapter are still completed. A developer who jumps into writing software without carefully thinking it through—often referred to as *cowboy coding*—ends up with errors, patches, and systems that are unreliable and difficult to update.

Figure 6-2 identifies the major models used for analysis and design. They're intended for object-oriented development, although they also share many characteristics with traditional development models, as explained in Online Chapter B. Notice that some model types are outputs of both analysis and design. For example, class and state machine diagrams appear on both sides of Figure 6-2. For class diagrams, some content, such as classes, attributes, and relationships,

FIGURE 6-2 Analysis and design models



is defined during analysis. Additional content, such as attribute data types, keys, indexes, and the class methods or functions, is added during design.

You should already be familiar with the analysis models shown on the left side of Figure 6-2. In the following chapters, you will learn how to create the design models shown on the right side of the figure.

■ Design Activities

Figure 6-3 identifies the activities of systems design. This section provides a short introduction to each of these design activities. In-depth explanation and instruction on the specific concepts and skills for each design activity are given later in the text.

Systems design involves specifying in detail how a system will work when deployed within a specific technology environment. Some of the details may have been defined during systems analysis, but much more detail is added during design. In addition, each part of the final solution is heavily influenced by the design of all the other parts. Thus, systems design activities are usually done in parallel. For example, the database design influences the design of application components, software classes and methods, and the user interface. Likewise, the technology environment drives many of the decisions for how system functions are distributed across application components and how those components communicate across a network. When an iterative approach to the SDLC is used, major design decisions are made in the first or second iteration; however, many decisions are revisited and refined during later iterations.

To better understand these design activities, you can summarize each one with a question. In fact, system developers often ask themselves these questions to help them stay focused on the objective of each design activity. **Figure 6-4** presents these questions:

FIGURE 6-3 *Design activities*

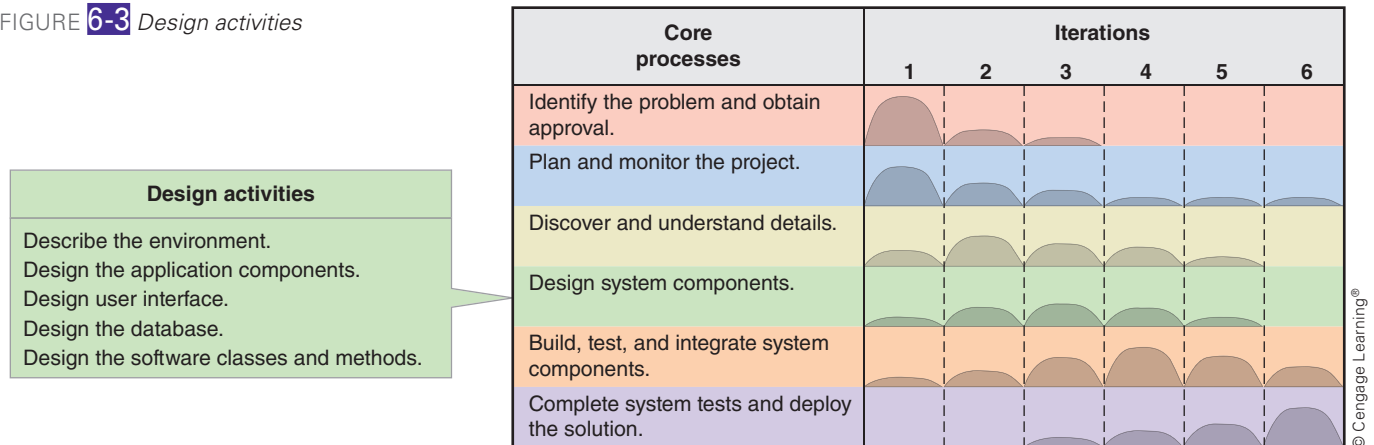


FIGURE 6-4 *Design activities and key questions*

Design activity	Key question
Describe the environment	How will this system interact with other systems and with the organization's existing technologies?
Design the application components	What are the key parts of the information system and how will they interact when the system is deployed?
Design the user interface	How will users interact with the information system?
Design the database	How will data be captured, structured, and stored for later use by the information system?
Design the software classes and methods	What internal structure for each application component will ensure efficient construction, rapid deployment, and reliable operation?

The following sections briefly discuss these design activities to better understand what is involved. In later chapters, you will develop the skills necessary for each of these activities.

■ Describe the Environment

For purposes of information systems design, the environment consists of two key elements:

- External systems
- Technology architecture

Interactions among external systems will usually have been identified and described by analysis activities. During analysis, those descriptions show how information flows to and from external systems. To design application components and their embedded software classes and methods, additional information about incoming and outgoing messages is needed, including:

- Precise message formats
- Web or network addresses of sources and destinations
- Communication protocols
- Security methods
- Error detection and recovery

The other key element, technology architecture, was defined in Chapter 2 as the set of computing hardware, network hardware and topology, and system software employed by an organization. For example, components of RMO's Consolidated Sales and Marketing System (CSMS) will be deployed on a set of computer servers, desktop computers, and mobile computing devices. Those devices will communicate across one or more networks employing multiple topologies and protocols. Each computing device will have an operating system (OS), communication capabilities, diverse input and output capabilities, and so forth. Additional system software, such as database management systems (DBMSs), Web server software, and firewalls will operate on some of the computer systems. The entire collection of hardware, system software, networks, and the yet-to-be-designed application components will form an integrated and comprehensive solution. Because the CSMS application software must interact with all of these elements once deployed, defining this environment in detail precedes other design activities.

Take a closer look at Figure 6-3, and note that all of the design activity names except the first begin with the word *design*. In contrast, the environment is *described* rather than designed. Why is this first activity different? The answer lies in the relative amount of control a designer has over the environment versus the other parts of the system. In general, the designer of a single system has little or no control over the environment. For example, with the RMO CSMS, the DBMS, user identity management system, and external systems (e.g., a UPS shipping system) are beyond the control of the designer. System components must be adapted to existing environmental elements, not the other way around.

Furthermore, although an organization does have control over its technology architecture, that control seldom extends to the designer of a single system. The technology architecture supports all of an organization's systems, so each of their designs is adapted to that architecture. Thus, from the point of view of a new system designer, the technology architecture is fixed, and the new system must be designed to efficiently interact with it. In some cases, a new system will require changes to existing technology architecture. In those cases, any changes to the technology architecture are considered and decided upon by the organization as a whole. Once those decisions are made, updating the architecture becomes the first step in designing the new system. Chapter 7 covers details of technology architecture and description of the technology environment.

application component a well-defined unit of software that performs one or more specific tasks

■ Design the Application Components

An **application component** is a well-defined unit of software that performs one or more specific tasks. Though that definition sounds simple, it hides complex details, including the following:

- **Size/scope.** A component could be small in size, like one method in a class or one script embedded within a Web page. Components can also be very large, like the order acceptance subsystem of an online store, a program containing thousands of lines of source code, or an entire Web site. Of course, application component size can also be between those extremes.
- **Programming languages.** Programs, functions, subroutines, and procedures are application components defined by traditional programming languages, such as C or FORTRAN. Object-oriented programming languages, such as Java and C#, use classes and methods as application components. Components written in scripting languages, such as JavaScript and PHP, are embedded within other application components such as Web pages. In short, the names, sizes, and relationships among application components vary widely from one programming language and toolkit to another.
- **Build or buy?** Some parts of an information system are written by an organization's information technology (IT) development staff. Others can be reused from other information systems owned by the organization, purchased separately, or acquired as part of a developer's toolkit. Other components might be deployed by an outside organization and made available via a Web services interface (e.g., Google Maps or a Federal Express package tracking service).

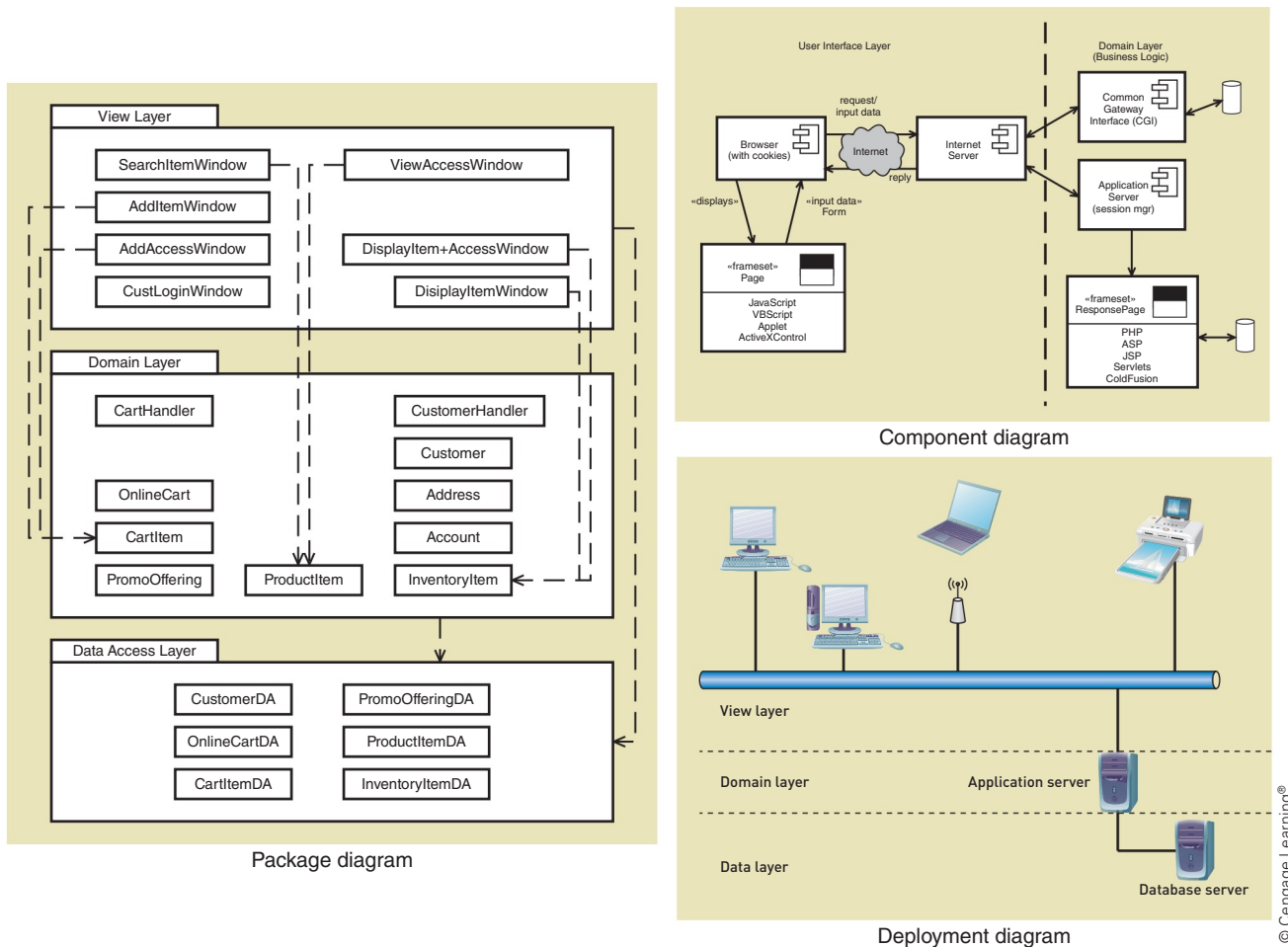
Key decisions made when designing application components include how functions of the system will be grouped or packaged, and how they'll interact with one another once built (or acquired) and assembled. One of the first steps in this design activity is separating the software into subsystems. Decisions are also made about the database infrastructure and about the multilayer design in which the user interface is separated from the business logic and database processing. The technology architecture will impact many of these design decisions. For example, an organization that employs Microsoft SQL Server and .NET will divide the application into components that are easily developed and deployed within that environment. A different organization that employs ORACLE DBMSs and related technology architecture will define subsystems that match that environment, usually with significant differences from a .NET design and deployment.

Application component design precedes the later design of software classes. In essence, application component design divides the functions of the system into smaller packages, each of which must be further specified in terms of software classes and methods. The result of application component design is a set of models (diagrams), often supplemented with other documents that provide additional detail. Examples of models produced by this design activity are shown in **Figure 6-5**. Chapters 7 and 13 cover the concepts and techniques that underlie these models.

■ Design the User Interface

To the user of a system, the user interface *is* the system. The quality of the user interface can make or break a system after deployment. Poor-quality interfaces deployed to customers and other external users often determine the failure of an entire business; high-quality user interfaces can be a competitive advantage. The reaction of employees to a new system interface can also determine system and business success. High-quality interfaces improve employee productivity and morale; poor-quality interfaces lead to errors and inefficiency.

FIGURE 6-5 Models produced during the activity: Design the application components

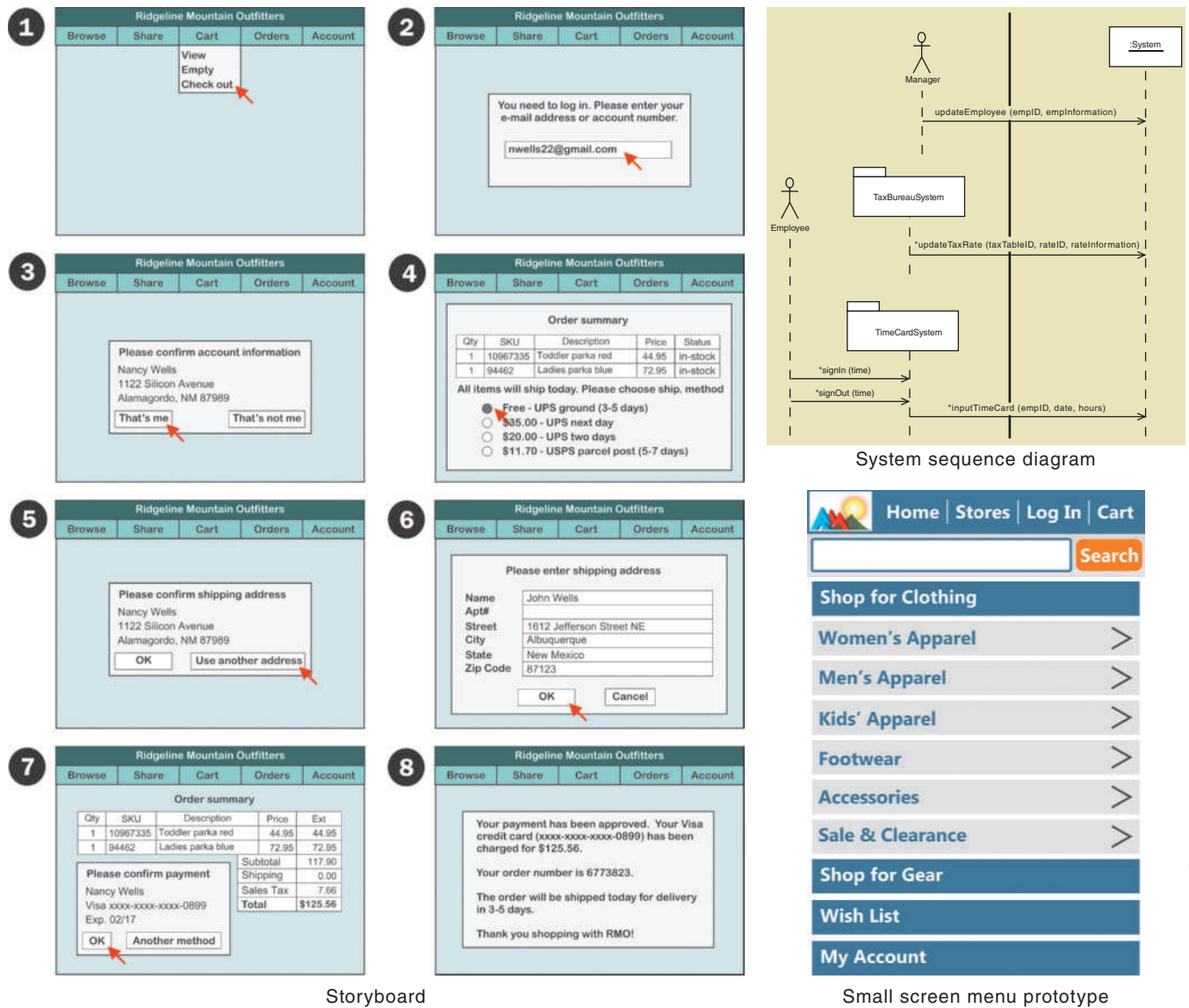


Ever-improving technology expands and changes user-interface requirements. A decade ago, user-interface design was concerned primarily with simplifying the presentation of information on desktop computer monitors and minimizing task-related keystrokes and mouse clicks. In the modern world, designers must deal with multiple screens ranging from phones to large multi-monitor displays. Multitouch screens, voice recognition, and built-in cameras that follow a user’s eye movements create new options and provide greater challenges for user-interface designers. Users might interact with a system using one technology at their desk, another in a conference room, and yet another while traveling. Designing a single user interface is now the exception, not the norm. Thus, the term *user interface* should generally be considered plural rather than singular.

Designing user interfaces can be thought of as both an analysis and design activity. Analysis-related aspects include understanding the user’s needs; how the user carries out his or her job; and the locations, devices, and physical contexts of user-system interactions. Design-related aspects include the distribution and packaging of related software and determining which device capabilities and embedded software will be incorporated into an interface.

Many types of models and tools are used to create a user-interface design, including mock-ups, storyboards, graphical layouts, and prototyping with screen-modeling tools (see Figure 6-6). System sequence diagrams developed during analysis are often reused and expanded during design to document user

FIGURE 6-6 Models produced during the activity: Design the user interface



Storyboard

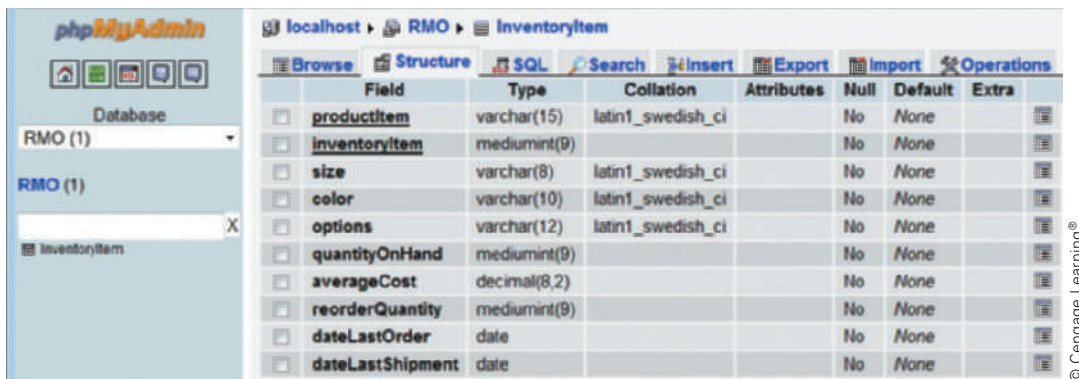
Small screen menu prototype

interaction with the system and its interface in detail. Chapter 8 describes many of the tools and techniques used to effectively build the user-interface design.

■ Design the Database

An integral part of every computer information system is the information itself, with its underlying database. The data model (the domain model class diagram) is created early during systems analysis and is then used to create the implementation model of the database. Usually, the first decision is to determine the database structure. Sometimes, the database is a collection of traditional computer files. More often, it is a relational database consisting of dozens or even hundreds of tables that interact with each other. Sometimes, files and relational databases are used in the same system. Another decision that needs to be made is whether the database should be centralized or distributed. The internal properties of the database must also be designed, including such things as attribute types, default values, and access controls. **Figure 6-7** is an example of an RMO database table definition for inventory items in MySQL.

FIGURE 6-7 Sample database table definition in MySQL



Field	Type	Collation	Attributes	Null	Default	Extra
<input type="checkbox"/> productItem	varchar(15)	latin1_swedish_ci		No	None	
<input type="checkbox"/> inventoryItem	mediumint(9)			No	None	
<input type="checkbox"/> size	varchar(8)	latin1_swedish_ci		No	None	
<input type="checkbox"/> color	varchar(10)	latin1_swedish_ci		No	None	
<input type="checkbox"/> options	varchar(12)	latin1_swedish_ci		No	None	
<input type="checkbox"/> quantityOnHand	mediumint(9)			No	None	
<input type="checkbox"/> averageCost	decimal(8,2)			No	None	
<input type="checkbox"/> reorderQuantity	mediumint(9)			No	None	
<input type="checkbox"/> dateLastOrder	date			No	None	
<input type="checkbox"/> dateLastShipment	date			No	None	

Analysts must consider many important technical issues when designing the database. Many of the technical (as opposed to functional) requirements defined during systems analysis concern database performance needs (such as response times). Much of the design work might involve performance tuning to make sure the system actually responds quickly enough. Security and encryption issues, which are important aspects of information integrity, must be addressed and designed into the solution. Given today's widespread connectivity, a database may need to be replicated or partitioned at various locations around the world. It is also not uncommon to have multiple databases with distinct DBMSs. These databases may be distributed across multiple database servers and may even be located at completely different sites. These highly technical issues often require specialized skills from experts in database design, security, performance, and physical configuration. A final key aspect of database design is making sure the new databases are properly integrated with existing databases. Chapter 9 describes database design in detail.

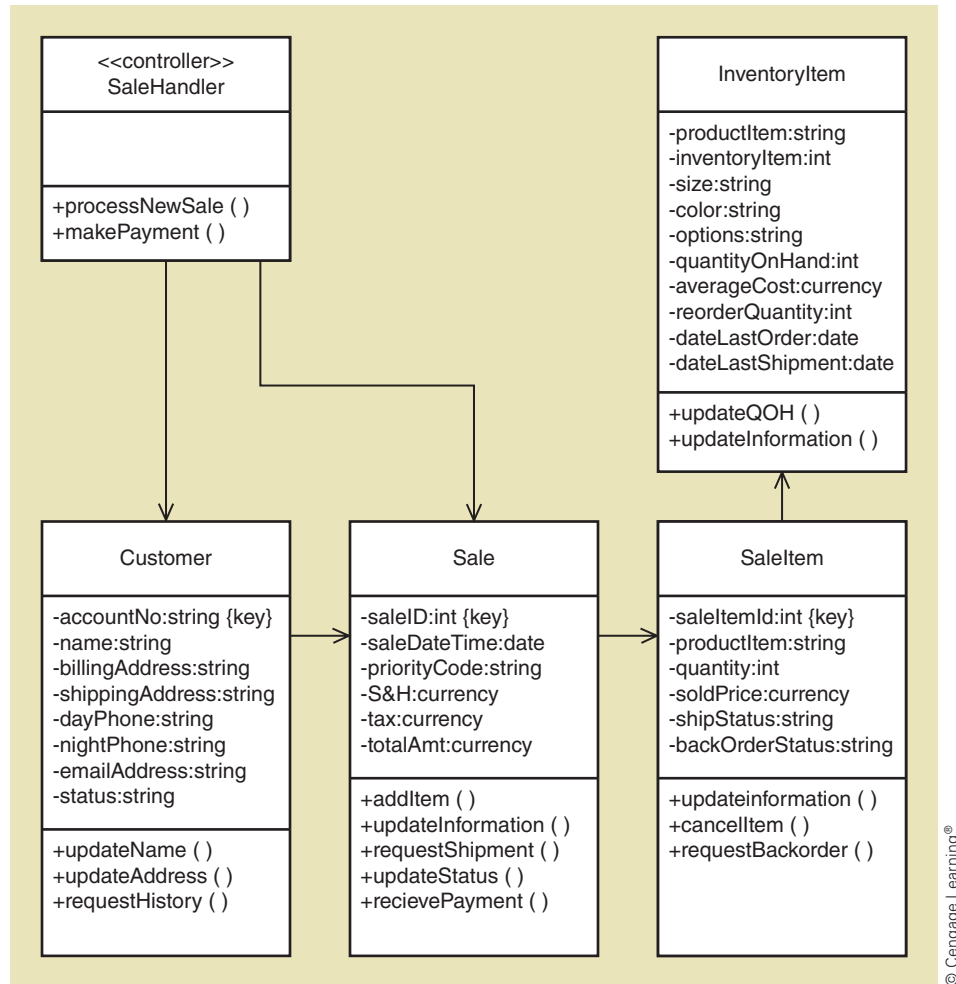
■ Design the Software Classes and Methods

During this design activity, analysis models, including class diagrams, system sequence diagrams, and state-machine diagrams, are extended to incorporate software-specific elements. Designers also create additional models such as sequence diagrams. These models are blueprints for software methods that will be programmed, tested, and eventually deployed. The purpose of this activity is to describe in enough detail how the software programs work, so that programmers can write the code for the functions and methods. **Figure 6-8** shows one of the most important models generated during this activity: a sample design class diagram for RMO's CSMS system. Chapters 12, 13, and 14 explain the details of designing software classes and methods, testing them, and packaging them for deployment.

■ System Controls and Security

Modern systems are subject to a variety of risks, ranging from malfunction due to incorrect data input, inadvertently revealing confidential information, and malicious destruction by outside parties. The terms *control* and *security* are blanket terms for the wide variety of ways that system designers and operators mitigate such risks. Some controls and security methods are embedded within the technology architecture. For example, most organizations have a unified method for identifying authorized users and providing them (and only them) with access to the data and applications they need to do their jobs, using identity

FIGURE 6-8 Partial design class diagram for RMO's CSMS



management and access controls. These controls are services shared across an entire organization's portfolio of applications.

Other controls and security are system-specific, and must be specifically designed by that system's developers. For example, a payroll-processing system typically includes significant controls to ensure that only valid employees or supervisors submit payroll information, that correct computations are performed on valid data inputs, and to define which data and processing results can be accessed by which users. Additional controls are part of the system's database and are designed into its software methods. System-specific controls often interact with organization-wide controls and security measures. For example, an application that enables employees to view and modify their own personal payroll information generally relies on organization-wide identity management methods, such as usernames and passwords.

The importance of controls and security to modern systems might lead you to believe that their design should be a security-specific design activity and should be listed in Figure 6-3. Doing so would be consistent with security's importance, but it would also obscure the fact that security and controls aren't separate and distinct from other design activities. Rather, they are issues that need to be embedded within all of the other design activities. That is, controls and security are important parts of other design activities, such as designing a user interface, a database, or application components. Because they are embedded in all other design activities, this chapter discusses key aspects so the concepts can be reused in the later design chapters.

■ Designing Integrity Controls

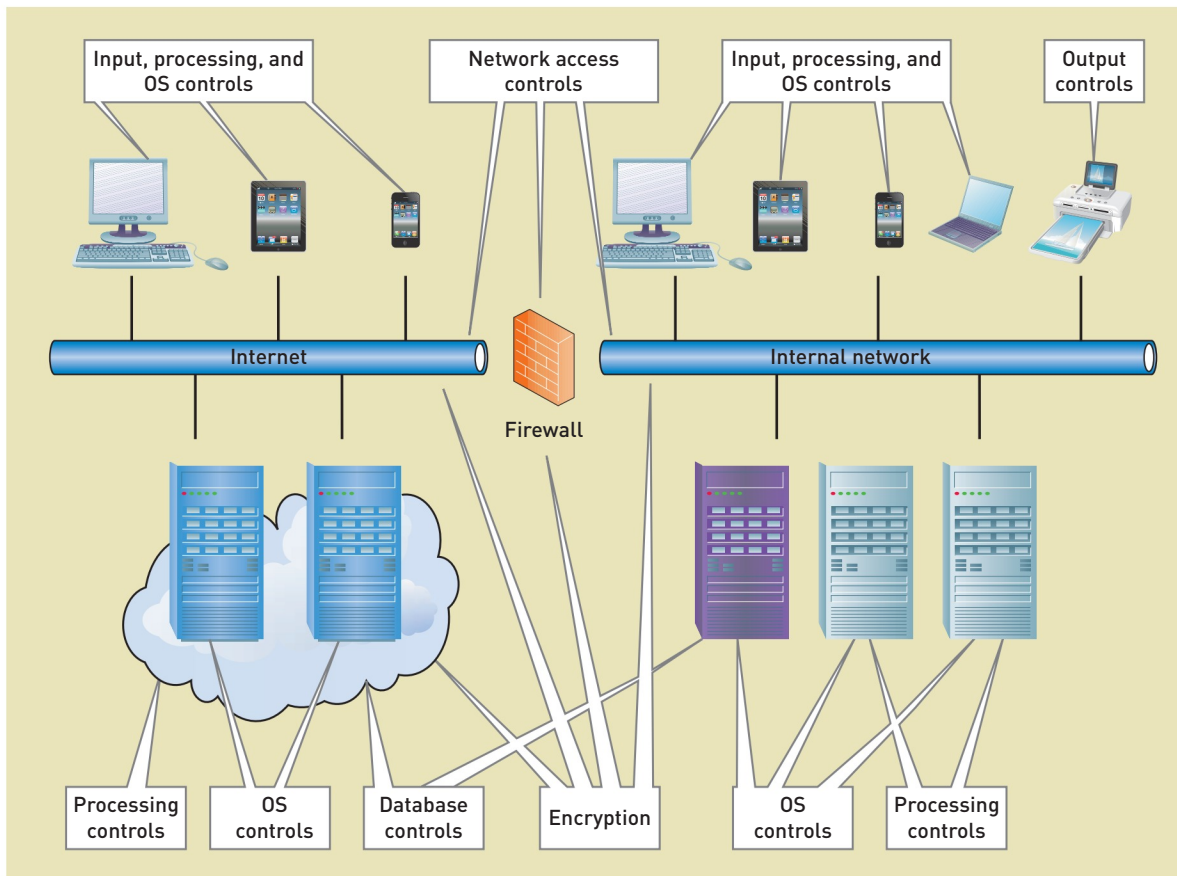
Controls are mechanisms and procedures that are built in to a system to safeguard the system and the information within it. Here are a few scenarios that illustrate the need for controls:

- A furniture store sells merchandise on credit with internal financing. Salespeople sometimes sell furniture on credit to friends and relatives. How do we ensure that only authorized employees can extend credit and record payments and adjustments to credit accounts?
- A bookkeeper uses accounting software to generate electronic payments to suppliers. How does the system ensure that the payment is for goods or services that were actually received? How does the system ensure that no one can generate payments to a bogus supplier?
- An online retailer collects and stores credit card and other information about customers. How does the company ensure that customer data is protected and secure?

As shown in **Figure 6-9**, controls are incorporated into various parts of the system. Some of the controls—called **integrity controls**—must be integrated into both the application programs that are being developed and the database that supports them. Other controls—usually called *security controls*—are part of the operating system and the network. Integrity controls ensure correct system function by rejecting invalid data inputs, preventing unauthorized data outputs, and protecting data and programs against accidental or malicious tampering. Security controls tend to be less application-specific. The distinction between

integrity controls controls that reject invalid data inputs, prevent unauthorized data outputs, and protect data and programs against accidental or malicious tampering

FIGURE 6-9 Security and integrity control locations



© Cengage Learning®

the two isn't precise because there is some overlap and because designers typically use both types. This section explains integrity controls. Later sections discuss security controls.

The primary objectives of integrity controls are as follows:

- To ensure that only appropriate and correct business transactions occur
- To ensure that the transactions are recorded and processed correctly
- To protect and safeguard the assets of the organization (including hardware, software, and information)

As described in the following subsections, organizations incorporate many types of controls into their systems and databases to achieve these objectives. Each control type addresses such specific issues as accuracy of inputs and limiting access to authorized users. No control type is sufficient by itself to achieve all the objectives; thus, a layered and multifaceted system of controls is required.

■ Input Controls

input controls controls that prevent invalid or erroneous data from entering the system

Input controls prevent invalid or erroneous data from entering the system. Input controls can be applied to data entered by people or data transmitted from internal or external systems. Input controls can be implemented within application programs, the database schema, or both. Commonly used input control types include:

value limit controls controls that check numeric data input to ensure that the value is reasonable

- **Value limit controls** check numeric data inputs to ensure that the amount entered is reasonable. For example, the amount of a sale or the amount of a commission usually falls within a certain range of values. A control might reject negative values or those that exceed a certain threshold, such as \$10,000.

completeness controls controls that ensure that all required data values describing an object or transaction are present

- **Completeness controls** ensure that all required data values describing an object or transaction are present. For example, when a shipping address is entered, the system might check whether enough information has been provided for successful delivery.

data validation controls controls that ensure that numeric fields that contain codes or identifiers are correct

- **Data validation controls** ensure that numeric fields containing codes or identifiers are correct. For example, a program for entering letter grades for a course might check that entered values match a set of predefined valid grades (such as A, B, C, D, and F) and reject any other entries.

field combination controls controls that review combinations of data inputs to ensure that the correct data are entered

- **Field combination controls** review various combinations of data inputs to ensure that the correct data are entered. For example, on an insurance policy, the application date must be prior to or the same as the effective date of policy coverage.

■ Output Controls

output controls controls that ensure that output arrives at the proper destination and is accurate, current, and complete

Systems outputs come in various forms, including data output that is used by other systems, printed reports, and data displayed on traditional or mobile computer displays. **Output controls** ensure that output arrives at the proper destination and is accurate, current, and complete. It is especially important that outputs with sensitive information arrive at the proper destination and that they cannot be accessed by unauthorized persons. Common types of output controls include:

- **Physical access controls to printers.** Printers and printed outputs should be located in secure areas accessible only by authorized personnel.
- **Discarded output control.** Although often ignored during systems design, physical control of discarded printed outputs containing sensitive data is a must because *dumpster diving* is an effective way to access data without authorization. Sensitive printed documents should be segregated from other trash and shredded or burned.

- **Access controls to programs that display or print.** Program access controls restrict which users can access specific programs and program functions, usually via such a mechanism as a username and password. In some instances, a system designer might restrict program or function access by access device. This extra safeguard is used primarily for military or other systems that house workstations in secure areas and provide access to the system's information to anyone who has access to the area.
- **Formatting and labeling of printed outputs.** System developers ensure completeness and accuracy by printing control data on the output report. For example, every report should have a date and time stamp—for the time the report was printed and for the date(s) of the underlying data. To ensure that a document is complete, designers typically incorporate such formatting features as pagination in the “page ____ of ____” format, control totals, and an “end of report” trailer.
- **Labeling of electronic outputs.** Electronic outputs typically include internal labels or tags that identify their source, content, and relevant dates. They may also include control totals or checksums that enable the recipient to determine whether content has been lost or altered.

■ Redundancy, Backup, and Recovery

Redundancy, backup, and recovery procedures are designed to protect software and data from hardware failure and from such catastrophes as fire, flood, and malicious destruction. Most operating systems and DBMSs incorporate support for all three. Many organizations that need continuous access to their data and systems employ redundant servers, databases, and sites. Each server or site hosts copies of application software, files, databases, and other important resources. Updates made to one server or site are immediately or frequently synchronized with the other copies to ensure consistency. If one site or server fails, the others are still accessible, and the organization continues to function.

Backup procedures make partial or full copies of a file system (data and software) or a database to removable storage media, such as magnetic tape, or to data storage devices or servers at another site. Unlike redundant sites or servers, backup copies stored off-site can't be accessed directly by application software or users. Instead, recovery procedures read the off-site copies and replicate their contents to a server that can then provide access to programs and users. Backup and recovery operations can take from minutes to hours. Backups are typically scheduled during periods of low utilization. Recovery is performed when needed, and the database is unavailable until the recovery procedures are completed.

■ Fraud Prevention

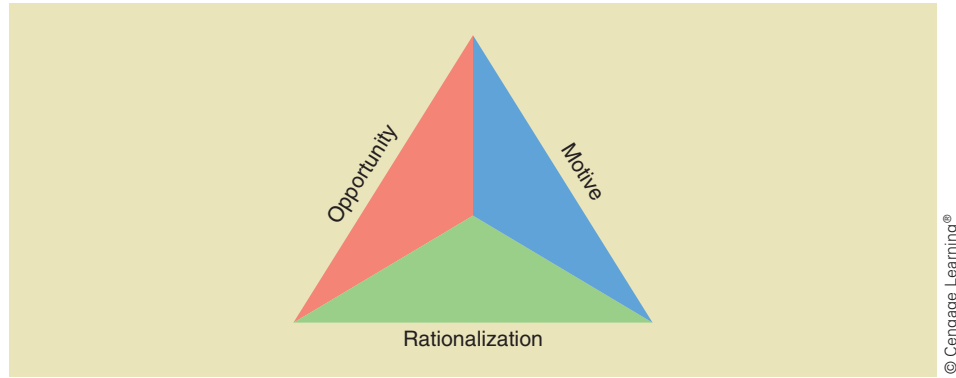
System designers and administrators are rightly concerned with security breaches arising from outside the organization. But they often pay inadequate attention to an equally serious problem: the use of the system by authorized users to commit fraud. Obviously, integrity and security controls won't completely eliminate fraud; however, system developers should be aware of the fundamental elements that make fraud possible and incorporate controls to combat it.

In the 1950s, fraud researchers developed a widely used model called the **fraud triangle** (see **Figure 6-10**). Much as a fire needs fuel, heat, and oxygen to burn, committing fraud requires three elements:

- **Opportunity.** The ability of a person to take actions that perpetrate a fraud. For example, unrestricted access to all functions of an accounts payable system enables an employee to generate false vendor payments.

fraud triangle a model of fraud that states that opportunity, motivation, and rationalization must all exist for a fraud to occur

FIGURE 6-10 The fraud triangle



- **Motivation.** A desire or need for the results of the fraud. Money is the usual motivation, although a desire for status or power, or even the need to be a “team player” may be contributing factors.
- **Rationalization.** An excuse for committing the fraud or an intention to “undo” the fraud in the future. For example, an employee might falsify financial reports to stave off bankruptcy, thus enabling fellow workers to keep their jobs. Or an employee might steal money to pay a gambling debt or medical bills, with the intention of repaying the money later.

System designers have little or no control over motive and rationalization, but they can minimize or eliminate opportunity by designing and implementing effective controls. **Figure 6-11** contains several of the more important factors that increase the risk of fraud. This list isn’t comprehensive, but it does provide a foundation developers can use to design a computer system that reduces the opportunity for fraud. As a system developer, you should include discussions with your users and project teams to ensure that adequate controls have been included to reduce fraud.

security controls controls that protect the assets of an organization from all threats, with a primary focus on external threats

■ Designing Security Controls

Although the objective of **security controls** is to protect the assets of an organization from all threats, the primary focus is on external threats. In addition

FIGURE 6-11 Fraud risks and prevention techniques

Factors affecting fraud risk	Risk-reduction techniques
Separation of duties	Design systems so those with asset custody have limited access to related records. Also, ensure that no one has sufficient system access to commit and cover up a fraud.
Records and audit trails	Record all transactions and changes in asset status. Log all changes to records and databases, and restrict log access to a few trusted persons.
Monitoring	Incorporate regular and systematic procedures to review records and logs for unusual transactions, accesses, and other patterns.
Asset control and reconciliation	Limit physical access to valuable assets, such as inventory, and periodically reconcile physical asset counts with related records.
Security	Design security features into individual systems and supporting infrastructure. Review and test security features frequently. Use outside consultants to conduct penetration testing attack and fraud vectors from external and internal sources.

to the objectives enumerated earlier for integrity controls, security controls have two further objectives:

- Maintain a stable, functioning operating environment for users and application systems (usually 24 hours a day, 7 days a week).
- Protect information and transactions during transmission across insecure environments such as public wireless networks and the Internet.

The first objective—to maintain a stable operating environment—focuses on security measures to protect the organization’s systems from external attacks from hackers, viruses, and worms, as well as denial-of-service attacks. The most common security control against such risks is a firewall installed between internal systems and the Internet (see Figure 6-9).

The second objective—to protect transactions during transmission—focuses on the information that is sent or received via the Internet. Once a transaction is sent outside the organization, it could be intercepted, destroyed, or modified. Thus, designers define security controls that protect data in transit from the source to the destination.

The most common security control points are network and computer operating systems because they exercise direct control over such assets as files, application programs, and disk drives. All modern operating systems contain extensive security features that can identify users, restrict access to files and programs, and secure data transmission among distributed software components. Because operating system security is the foundation of security for most information systems, a key task in defining security controls is determining which operating system security features will be enabled and how they will be configured.

■ Access Controls

An **access control** limits a user’s ability to access resources, such as servers, files, Web pages, application programs, and database tables. Operating systems, networking software, and DBMSs all provide access control systems, and all of these can be configured to share a common access control system. Access control systems rely on these common elements:

- **Authentication** is the process of identifying users who request access to sensitive resources. Users can be authenticated through methods (or factors) such as usernames and passwords, smart cards, challenge questions and responses, or biometric methods, including fingerprint and retinal scans, or voice recognition. **Multifactor authentication** uses multiple methods for increased reliability.
- An **access control list** is a list attached or linked to a specific resource that describes users or user groups and the nature of permitted access (e.g., read data, update data, or execute program). Users who don’t appear in the access control list can’t use the associated resource.
- **Authorization** is the process of allowing or restricting a specific authenticated user’s access to a specific resource based on an access control list.

To build an effective access control system, a designer must categorize system users and determine what type(s) of access each resource requires. **Figure 6-12** illustrates three user categories or types and the role of the access control system in allowing or restricting their access:

- **Unauthorized users** are people who aren’t allowed access to any part or function of the system. Such users include employees who are prohibited from accessing all of the system, former employees who are no longer permitted to access the system, and outsiders, such as hackers and intruders.
- **Registered users** are those who are authorized to access the system. Normally, types of registered users are defined depending on what they are authorized to view and update. For example, some users may be allowed

access control a control that limits a user’s ability to access resources, such as servers, files, Web pages, application programs, and database tables

authentication the process of identifying users who request access to sensitive resources

multifactor authentication the process of using multiple authentication methods for increased reliability

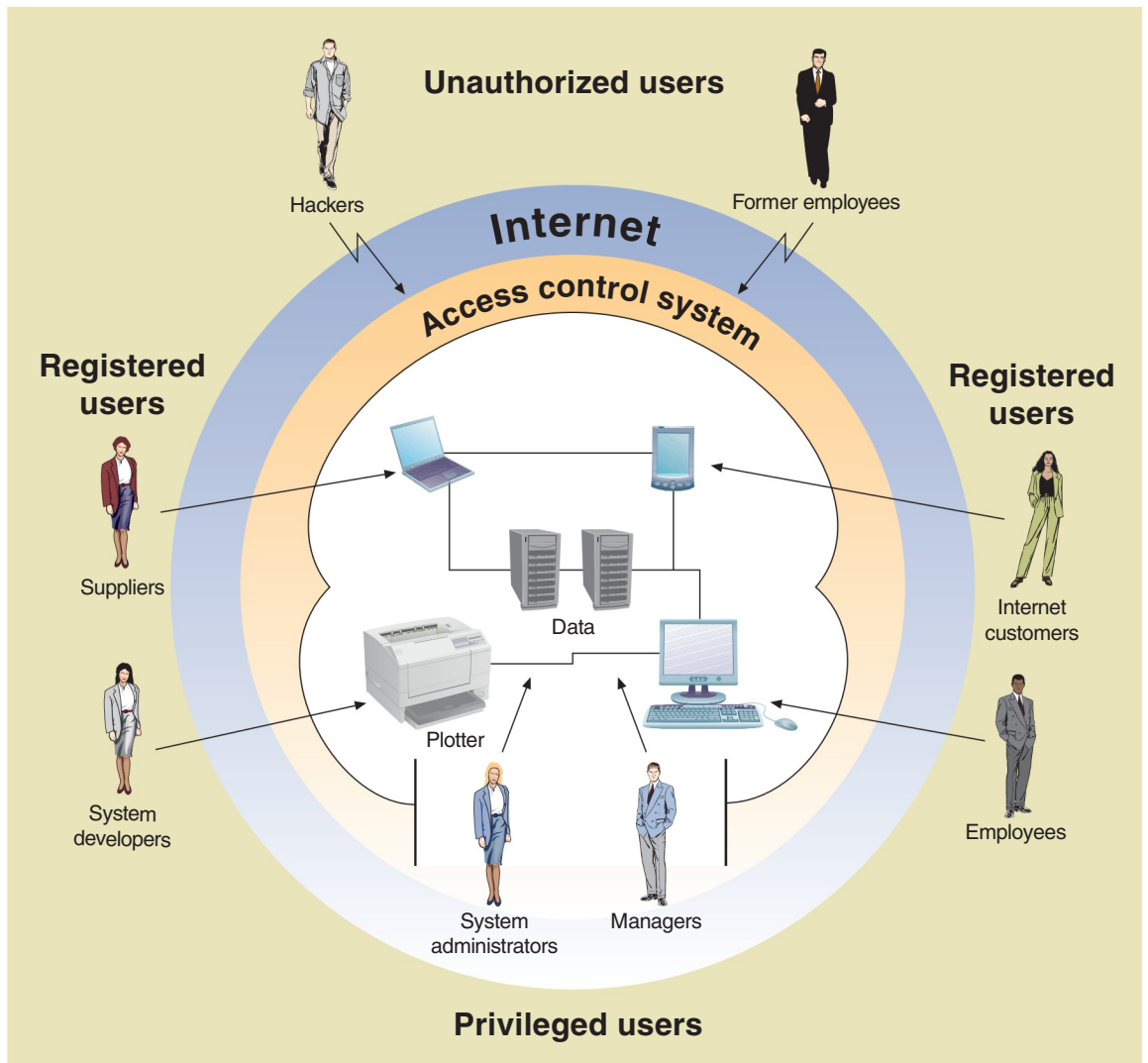
access control list a list attached or linked to a specific resource that describes users or user groups and the nature of permitted access

authorization the process of allowing or restricting a specific authenticated user’s access to a specific resource based on an access control list

unauthorized users people who aren’t allowed access to any part or function of the system

registered users people who are authorized to access the system

FIGURE 6-12 Users and their access to computer systems



© Cengage Learning®

privileged users people who have access to the source code, executable program, and database structure of the system

- **Privileged users** are people who have access to the source code, executable program, and database structure of the system, including system programmers, application programmers, operators, and system administrators. These people may have differing levels of security access.

■ Data Encryption

No access control system is perfect, so designers must anticipate access control breaches and provide other measures to protect the confidentiality of data. Protective measures must also be applied to data that is stored or transmitted outside the organization's own network, such as transaction data sent by remote suppliers or customers, and interactions between internal applications and cloud service providers. Common types of information that require additional protection include:

- Financial information
- Personal information, such as credit card numbers, bank account numbers, payroll information, or health-care information

- Strategies and plans for products and other mission-critical data
- Government and sensitive military information
- Data stored on such portable devices as laptop computers and cell phones

encryption the process of altering data so unauthorized users can't view them

decryption the process of converting encrypted data back to their original state

encryption algorithm a complex mathematical transformation that encrypts or decrypts binary data

encryption key a binary input to the encryption algorithm—typically a long string of bits

symmetric key encryption an encryption method that uses the same key to encrypt and decrypt the data

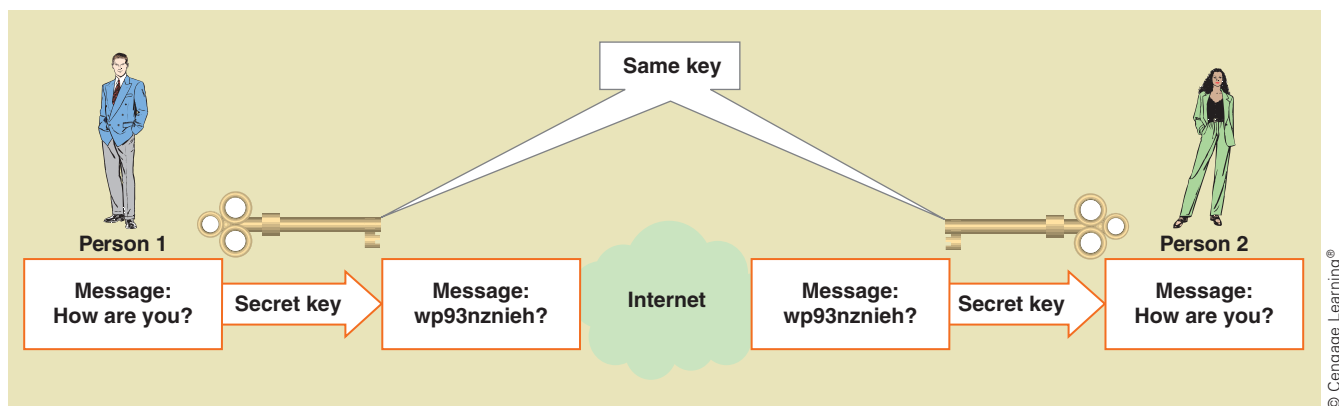
Encryption is the primary method of securing data within internal systems and during transmission. **Encryption** is the process of altering data so unauthorized users can't view them. **Decryption** is the process of converting encrypted data back to their original state. Data stored in files, or databases on hard drives or other storage devices, can be encrypted to protect against theft. Data sent across a network can be encrypted to prevent eavesdropping or theft during transmission: A thief or eavesdropper who steals or intercepts encrypted data receives a meaningless group of bits that are difficult or impossible to convert back into the original data.

An **encryption algorithm** is a complex mathematical transformation that encrypts or decrypts binary data. An **encryption key** is a binary input to the encryption algorithm—typically a long string of bits. The encryption algorithm varies the data transformation based on the encryption key, so data can be decrypted only with the same key or a compatible decryption key. Many encryption algorithms are available, and a few—including Data Encryption Standard (DES) and several algorithms developed by RSA Security—are widely deployed governmental or Internet standards. An encryption algorithm must generate encrypted data that are difficult or impossible to decrypt without the encryption key. Decryption without the key becomes more difficult as key length is increased, and sender and receiver must use the same or compatible algorithms.

Figure 6-13 shows an example of **symmetric key encryption**, in which the same key encrypts and decrypts the data. A significant problem with symmetric key encryption is that sender and receiver use the same key, which must be created and shared securely. Security is compromised if the key is transmitted over the same channel as messages encrypted with the key. Further, sharing a key among many users increases the possibility of key theft. Because of these risks, symmetric key encryption is frequently used with data stored in files and databases, but is used only in conjunction with asymmetric encryption to transmit data over a network.

Data stored by file and database servers can be encrypted with symmetric key encryption to protect against unauthorized access that bypasses the OS or DBMS to directly access the physical data store. Typically, the encryption key is stored on a different server that is queried via secure login when the server first boots up. Data stored on laptops and other portable devices are often encrypted to protect against unauthorized access due to loss, theft, or disposal.

FIGURE 6-13 Symmetric key encryption



remote wipe a security measure that automatically deletes sensitive data from a portable device when unauthorized accesses are attempted

asymmetric key encryption an encryption method that uses different keys to encrypt and decrypt the data

public key encryption a form of asymmetric key encryption that uses a public key for encryption and a private key for decryption

An additional security measure for portable devices is a technique commonly called **remote wipe**, which automatically deletes sensitive data from portable devices under certain conditions, such as repeated failure to enter a valid username and password, or an attempt to access a database or file from an unauthorized application. Remote wipe is commonly used with apps on portable devices that sync sensitive data from a server. An unauthorized synchronization attempt triggers the remote wipe via a command from the app or the server.

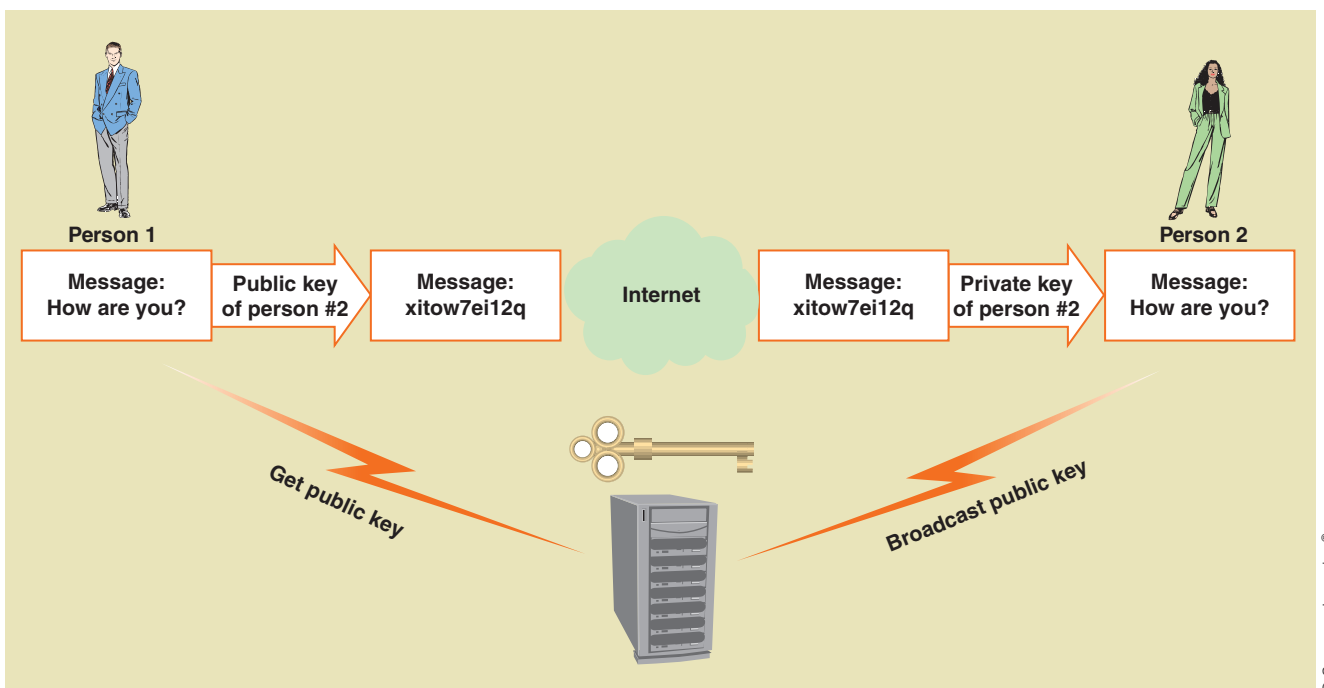
Asymmetric key encryption uses different, but compatible keys to encrypt and decrypt data. **Public key encryption** is a form of asymmetric key encryption that uses a public key for encryption and a private key for decryption. The two keys are like a matched pair. Once information is encrypted with the public key, it can be decrypted only with the private key. It can't be decrypted with the same public key that encrypted it. Organizations that use this technique broadcast their public key so it is freely available to anybody who wants it. For example, a company transmitting shipment data to a shipper can access the public key from the shipper's Web site and encrypt the shipment information before transmission. The shipper then decrypts the message with the private key. Because no one else has the private key, no one else can decrypt the message.

Some asymmetric encryption methods can encrypt and decrypt messages in both directions. That is, in addition to using the public key to encrypt a message that can be decrypted with the private key, an organization can also encrypt a message with the private key and decrypt it with the public key. Notice that both keys must still work as a pair, but the message can go forward or backward through the encryption/decryption pair. This second technique is the basis for digital signatures and certificates, which are explained in the next section. **Figure 6-14** illustrates an asymmetric key encryption transmittal.

■ Digital Signatures and Certificates

Encryption is an effective technique for a secure exchange of information between two entities who have appropriate keys. However, how do you know that the

FIGURE 6-14 Asymmetric key encryption



digital signature a technique in which a document is encrypted by using a private key to verify who wrote the document

digital certificate an institution's name and public key (plus other information, such as address, Web site URL, and validity date of the certificate) encrypted and certified by a third party

certifying authorities widely accepted issuers of digital certificates

entity on the other end of the communication is really who you think it is? A **digital signature** is a technique in which a message or document is encrypted with a private key and decrypted with the public key. For example, your bank might encrypt a message sent to you with its private key. If you decrypt the message with your bank's public key and the message is readable, then you know the message was sent by your bank. If you decrypt the message with the public key and the result is gibberish, then you know the message was *not* sent by your bank. Encoding a message with a private key is called *digital signing*.

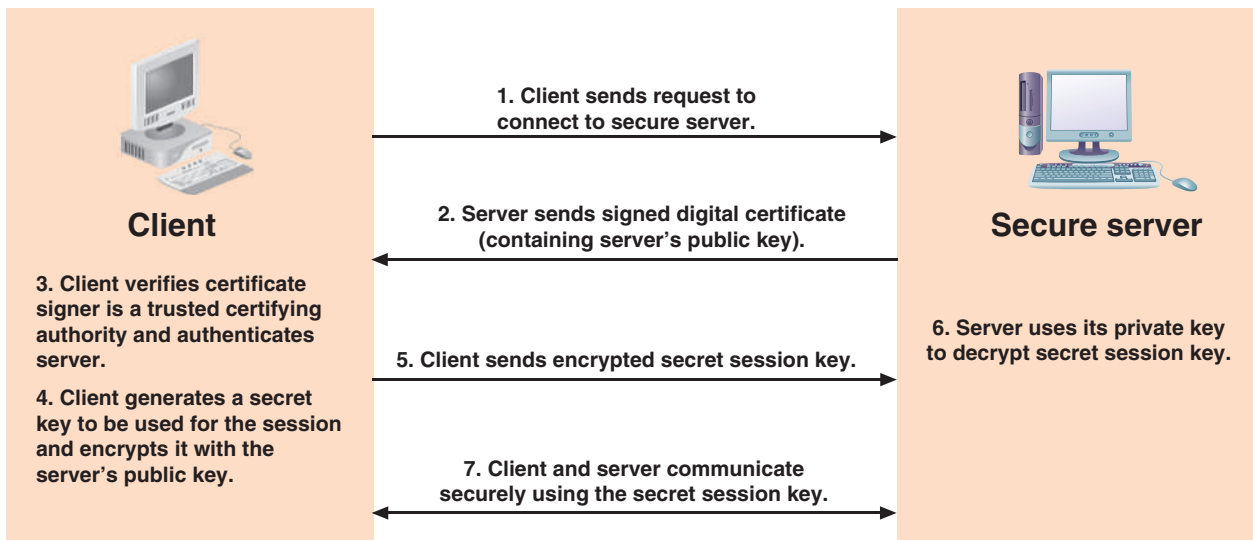
Taking the example one step further, you can ask the question, "How do I know that the public key I have is the correct public key and not some counterfeit key?" In other words, maybe someone is impersonating another entity and passing out false public keys to intercept encoded messages (such as financial transactions) and steal information. In essence, the problem is ensuring that the key purported to be the public key of some institution is, in fact, that institution's public key. The solution to that problem is a certificate.

A certificate—or **digital certificate**—is an institution's name and public key (plus other information, such as address, Web site URL, and validity date of the certificate), encrypted and certified by a third party. Many third parties, such as VeriSign and Equifax, are very well known and widely accepted **certifying authorities**. In fact, they are so well known that their public keys are built right into the current browsers such as Firefox, Chrome, Safari, and Internet Explorer. As shown in Figure 6-15, you can know that the entities with whom you are communicating are, in fact, who they say they are, and that you do have their correct public key.

An organization or individual who wants a certificate with his or her name and public key buys a certificate from a well-known certifying authority such as VeriSign. The certifying authority encrypts the data with its own private key (signs the data) and gives the data back to the original entity. When someone, such as a customer, asks you for your public key, you send the certificate. The customer receives the certificate and opens it with the certifying authority's public key. Because the certifying authority is so well known, its public key is built in to everyone's browser and is essentially impossible to counterfeit. The customer can now be sure that he or she is communicating with you and can do so with encrypted messages by using your public key.

A variation of this scenario occurs when a buyer and seller transmit their certificates to one another. Each participant can decrypt the certificate by using the

FIGURE 6-15 Using a digital certificate



© Cengage Learning®

certifying authority's public key to extract such information as name and address. However, to ensure that the public key contained within the certificate is valid, the certificates are transmitted to the certifying authority for verification. The authority stores certificate data, including public keys, within its database and verifies transmitted certificates by matching their content against the database.

■ Secure Transactions

Secure electronic transactions require a standard set of methods and protocols that address authentication, authorization, privacy, and integrity. Netscape originally developed the **Secure Sockets Layer (SSL)** to support secure transactions. SSL was later adopted as an Internet standard and renamed **Transport Layer Security (TLS)**, although the original name is still widely used.

TLS is a method for secure message transmission over the Internet. Sender and receiver first establish a connection by using ordinary Internet protocols and ask each other to create a TLS connection. Sender and receiver then verify each other's identity by exchanging and verifying identity certificates as explained previously. At this point, either or both have exchanged public keys, so they can send secure messages. Because asymmetric encryption is quite slow and difficult, the two entities agree on a protocol and encryption method—usually a single-key encryption method. Of course, all the messages to establish a secure connection are sent by using the public key/private key combination. Once the encryption technique has been determined and the secret, single key has been transmitted, all subsequent transmission is done by using the secret, single key.

IP Security (IPSec) is a newer Internet standard for secure message transmission. IPSec is usually implemented by operating systems or within network hardware devices, which enables it to operate with greater speed. IPSec can replace or complement SSL. The two protocols can be used at the same time to provide an extra measure of security. IPSec supports more secure encryption methods than SSL.

Hypertext Transfer Protocol Secure (HTTPS) is an Internet standard for securely transmitting Web pages. HTTPS supports several types of encryption, digital signing, and certificate exchange and verification. All modern Web browsers and servers support HTTPS. It is a complete approach to Web-based security, although security is enhanced when HTTPS documents are sent over secure TLS or IPSec channels.

Secure Sockets Layer (SSL) a standard set of methods and protocols that address authentication, authorization, privacy, and integrity

Transport Layer Security (TLS) an Internet standard equivalent to SSL

IP Security (IPSec) an Internet standard for secure transmission of low-level network packets

Hypertext Transfer Protocol Secure (HTTPS) an Internet standard for securely transmitting Web pages

CHAPTER SUMMARY

Systems design is the process of organizing and structuring the components of a system to enable the construction of the new system. The design encompasses key parts of the system, including its environment, application components, user interfaces, database, and software classes and methods. A description or design activity is associated with each design element. Design activity inputs consist of the models that were built during analysis. The outputs of the design consist of a set of diagrams, or models, that describe the architecture of the new system and the detailed logic within various programming components.

Key design elements include application components, the user interface, the database, and software classes and methods. Each element is the focus of a separate design activity, though the timing and

ordering of activities varies from project to project. A separate activity also defines the environment within which the system will be built and operated. Environmental characteristics tend to limit design choices and flexibility.

Integrity controls operate within a specific system to reject invalid data inputs, prevent unauthorized data outputs, and protect data and programs against accidental or malicious tampering. Security controls cross multiple systems to protect assets of an organization from all threats, with a primary focus on external threats. Although integrity and security controls are important design elements, there is no separate activity to create them; control development is included in each of the other design activities to ensure a robust and reliable system.

KEY TERMS

access control	encryption	privileged users
access control list	encryption algorithm	public key encryption
application component	encryption key	registered users
asymmetric key encryption	field combination controls	remote wipe
authentication	fraud triangle	Secure Sockets Layer (SSL)
authorization	Hypertext Transfer Protocol Secure (HTTPS)	security controls
certifying authorities	input controls	symmetric key encryption
completeness controls	integrity controls	Transport Layer Security (TLS)
data validation controls	IP Security (IPSec)	unauthorized users
decryption	multifactor authentication	value limit controls
digital certificate	output controls	
digital signature		

REVIEW QUESTIONS

- How does the objective of systems analysis differ from the objective of systems design?
- What are the inputs to systems design? What are the outputs?
- List and briefly describe each design activity.
- Why is the environment *described* while other key design elements such as the user interface and database are *designed*?
- What models are developed during each design activity?
- On a project that uses iterations to develop the system, in which iteration does systems design begin? Explain why.
- What are the key elements of the environment described during design activities?
- List at least three examples of application components.
- Why is the singular form of the term *user interface* usually a misnomer?
- Designing security and controls impacts the design of which other elements?
- Compare and contrast integrity controls and security controls. Why isn't there a separate activity to design them?
- Explain four types of integrity controls for input forms. Which have you seen most frequently? Why are they important?
- What are the two primary objectives of security controls?
- List and briefly describe the three elements of the fraud triangle. Over which element can a system designer exercise the greatest control?
- Briefly define or describe authentication, access control lists, and authorization.
- How does single-key (symmetric) encryption work? What are its strengths? What are its weaknesses?
- What is the difference between HTTPS and HTTP?

PROBLEMS AND EXERCISES

- Discuss the technology architecture and deployment environment for information systems at your work or school with a knowledgeable person. What are the key elements of the environment? If you were to implement a new system for the organization, which of the environmental elements could you change? Which could you choose to use or not use?
- Pick a major online retailer (e.g., Amazon, Walmart, or Sears). Examine the default browser-based shopping interface from the following devices: a desktop computer with a large monitor, a tablet computer, and a smartphone. How and why do content and layout vary across the devices? Are there any device-specific technologies employed in any of the interfaces, such as voice recognition, multitouch gestures,

- and gaze (eye) detection? Is a shopping app available for any of the devices? If so, how does the app's user interface differ from browser-based shopping on the same device?
3. This chapter described various situations that emphasize the need for controls. In the first scenario presented, a furniture store sells merchandise on credit. Based on the descriptions of controls given in this chapter, identify the various controls that should be implemented in the furniture store system to ensure that corrections to customer balances are made only by someone with the correct authorization. In the second scenario illustrating the need for controls, an accounts payable clerk uses the system to write checks to suppliers. Based on the information in this chapter, what kinds of controls would you implement to ensure that checks are written only to valid suppliers, that checks are written for the correct amount, and that all payouts have the required authorization? How would you design the controls if different payment amounts required different levels of authorization?
 4. Examine the privacy policy (or privacy section of the user agreement) for a major online service provider such as Gmail, eBay, or Facebook. What are the implications of the privacy guarantees for controls and security? Briefly describe cost-benefit trade-off of the guarantees.
 5. Look on the Web site you use to access your bank account(s). What kinds of security and controls are integrated into the system?
 6. Examine the information system of a local business, such as a fast-food restaurant, doctor's office, video store, grocery store, etc. Evaluate the screens (and reports, if possible). What kinds of integrity controls are in place? What kinds of improvements would you make?
 7. Search the Web for information about Pretty Good Privacy. What is it? How does it work? Find information about a passphrase. What does it mean? Start your research at <http://www.pgpi.org>.

CASE STUDY

County Sheriff Mobile System for Communications (CSMSC)

Law enforcement agencies thrive on information. In previous eras, it was sufficient to receive information through the police dispatch radio. Today, much more than voice-based information is required. Officers often need to check vehicle registrations, personal identities, outstanding warrants, mug shots, maps, and the locations of other officers.

One major difficulty with meeting this need for more information is figuring out how to transmit the data to remote and mobile locations. Local police agencies are sometimes able to restrict their transmission needs to

within the city limits. However, county sheriffs and state troopers often have to travel to remote locations that aren't within a metropolitan area's boundaries.

Suppose a rural county sheriff's department has received a grant to upgrade its existing voice-only communication system. Among other requirements, the grant specifies that all communications must be protected against eavesdropping and unintended information disclosure.

Your assignment: Recommend specific controls to be applied to radio, cellular, and satellite transmissions. How will you ensure that only sheriffs and other authorized users can use the system?

RUNNING CASE STUDIES

Community Board of Realtors®

The Community Board of Realtors' Multiple Listing Service (MLS) will be a Web-based application with extensions to allow wireless interaction between the agents and their customers using cell phones, tablets, and other portable devices.

Review the functional and nonfunctional requirements you have developed for previous chapters. Then, for each of the five design activities discussed in this chapter, list some specific tasks required for describing the environment and designing

application components, user interfaces, the database, and software methods. How will you integrate the design of integrity and security controls

into those tasks? You may want to refer back to the Tradeshow System discussed in Chapter 1 for some design specifics.

The Spring Breaks 'R' Us Travel Service

The SBRU information system includes four subsystems: Resort relations, Student booking, Accounting and finance, and Social networking. The first three are purely Web applications, so access to those occurs through an Internet connection to a Web server at the SBRU home office. However, the Social networking subsystem has built-in chat capabilities. It relies on Internet access, as students compare notes before they book their travel reservations and as they chat while traveling. To function properly, the system obviously requires a wireless network at each resort. SBRU isn't responsible for installing or maintaining the resort wireless network;

they only plan to provide some design specifications and guidelines to each resort. The resort will be responsible for connecting to the Internet and for providing a secure wireless environment for the students.

1. For which subsystem(s) is(are) integrity and security controls most important? Why?
2. What data should be encrypted during transmission through resort wireless networks to SBRU systems? Does your answer change if students interact with SBRU systems using a cell phone (directly, or as a cellular modem)?

On the Spot Courier Services

Previous chapters have described the technological capabilities that Bill Wiley wants for servicing his customers. One of the problems that Bill has is that his company is very small, so he cannot afford to develop any special-purpose equipment or even sophisticated software.

Given this limitation, Bill's need for advanced technological capabilities comes at an opportune time. Equipment manufacturers are developing equipment with advanced telecommunications capabilities, and freelance software developers are producing software applications—many of which provide the capabilities that Bill needs. The one caveat is that because this will be a live production system, it needs to be reliable, stable, error-free, dependable, and maintainable.

Let us review some of the required capabilities of the new system, which has been described in previous chapters:

Customers

- Customers can request package pickup via the Internet.
- Customers can check the status of packages via the Internet.
- Customers can print mailing labels at their offices.

Drivers

- Drivers can view their schedules via a portable digital device while on their routes.
- Drivers can update the status of packages while on their routes.
- Drivers can allow customers to “sign” for packages that are delivered.

- The system “knows” where the driver is on his route and can send updates in real time.
- Drivers can accept payments and record them on the system.

Bill Wiley (management)

- Bill can record package pickups from the warehouse.
- Bill can schedule delivery/pickup runs.
- Bill can do accounting, billing, etc.
- Bill can access the company network from his home.

Given these requirements, do the following:

1. What kind of fraud is possible in this scenario? By the customer? By the truck driver? By collaboration between system users? What steps should Bill take to minimize the opportunity for fraud?
2. What kind of access controls should be put in place? For the customer? (Notice the customer has no financial transactions. Would you change your answer if the customer could also make payments online?) For the truck driver? For Bill? Are the typical userID and password sufficient for all three, or would you require more or less for each?
3. Research on the Web and find out what is required to purchase a digital certificate for a Web site. Explain what Bill would have to do to implement a secure site with HTTPS. Do you recommend that Bill secure his site with HTTPS and digital security? Why or why not?

Sandia Medical Devices

As described in previous chapters, the Real-Time Glucose Monitoring (RTGM) system will include processing components on servers and on mobile devices with data exchange via 3G and 4G phone networks. Users will include patients, physicians, nurses, and physician assistants. In the United States, the Health Insurance Portability and Accountability Act of 1996 (HIPAA) mandates certain responsibilities regarding the privacy and security of electronic protected health information (ePHI). The law applies to what are collectively called covered entities—that is, health plans, health-care clearinghouses, and any health-care providers who transmit health information in electronic form. More information can be obtained from the U.S. Department of Health and Human Services Web site (www.hhs.gov).

In general, covered entities should do the following:

- Ensure the confidentiality, integrity, and availability of all ePHI they create, receive, maintain, or transmit.
- Identify and protect against reasonably anticipated threats to the security or integrity of the information.
- Protect against reasonably anticipated, impermissible uses or disclosures of the information.
- Ensure compliance by their workforces.

Specifically, covered entities should implement policies, procedures, and technologies that do the following:

- Specify the proper use of and access to workstations and electronic media.
- Regard the transfer, removal, disposal, and reuse of electronic media to ensure appropriate protection of ePHI.

- Allow only authorized persons to access ePHI.
- Record and examine access and other activity in information systems that contain or use ePHI.
- Ensure ePHI isn't improperly altered or destroyed.
- Guard against unauthorized access to ePHI that is being transmitted over an electronic network.

Answer these questions in light of HIPAA requirements:

1. Does HIPAA apply to the RTGM system? Why or why not?
2. How should the system ensure data security during transmission between a patient's mobile device(s) and servers?
3. Consider the data storage issues related to a patient's mobile device and the possible ramifications if the device is lost or stolen. What measures should be taken to protect the data against unauthorized access?
4. Consider the issues related to health-care professionals accessing server data by using workstations and mobile devices within a health-care facility. How will the system meet its duty to record and examine access to ePHI? If a health-care professional uses a mobile device outside a health-care facility, what protections must be applied to the device and/or any data stored within it or transmitted to it?
5. Consider the issues related to wired and wireless data transmission between servers and workstations within a health-care facility. What security duties, if any, apply to transmissions containing ePHI? Does your answer change if the servers are hosted by a third-party provider?

FURTHER RESOURCES

Frederick P. Brooks, *The Design of Design: Essays from a Computer Scientist*. Addison-Wesley, 2010.

Michael E. Whitman and Herbert J. Mattord, *Management of Information Security*. Cengage, 2014.

Terry Winograd, *Bringing Design to Software*. ACM Press, 1996.

CHAPTER SEVEN

CHAPTER OUTLINE

- ▶ Anatomy of a Modern Information System
- ▶ Architectural Concepts
- ▶ Interoperability
- ▶ Architectural Diagrams
- ▶ Describing the Environment
- ▶ Designing Application Components

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- ▶ Explain architectural concepts that influence system design, including ubiquitous computing and software, components, protocols, interoperability, and distributed architectures
- ▶ Describe and draw location, network, and deployment diagrams
- ▶ Describe a system's environment by drawing architectural diagrams and answering key questions
- ▶ Design larger application components based on use cases and other analysis models

OPENING CASE TECHNOLOGY DECISIONS AT WYSOTRONICS INC.

As James Schultz walked down the hall toward a meeting with his staff, he thought about his current job. For a year now, Schultz had been the vice president and chief information officer for a medium-sized supplier of electronic components to several large electronics firms, including Samsung and Acer. James's company, Wysotronics Inc., had been in business for many years, but had recently been having some problems with its internal computer systems. James was hired to fix the problems.

Soon after starting this job, James discovered that most systems were functioning properly, but that the infrastructure was a hodgepodge of disjointed computers and networks. On the corporate side, there were accounting systems and human resource systems, both of which were desktop client/server systems hosted on a local network computer that resided in the Accounting Department.

Engineering had its own database and network computers, which hosted several sophisticated engineering systems with intensive computing requirements. The engineers' local desktop systems were the most recent, up-to-date equipment and software. The server was also high capacity, with a large data repository to house all the engineering documents and images.

Marketing and Sales also had their own systems hosted on their own network server, which was connected to the Internet. The sales staff worked closely with the manufacturing and assembly plants to ensure that deliveries were on time, and they were frequently on the road, visiting Wysotronics's clients. It was their job to ensure that clients were satisfied with schedules, deliveries, and product quality, and they wanted to be able to access the sales and production databases while they were on the road. Unfortunately, the servers they used weren't very stable and continually had problems.

Perhaps the biggest problem was the supply chain management system. Wysotronics had a large manufacturing plant, an assembly plant, and several suppliers that needed access to the inventory and supply chain system. The current infrastructure didn't have enough capabilities to provide timely information to these facilities and suppliers.

Today's meeting objective was to plan and configure the total infrastructure of corporate systems. As he walked into the room, James was greeted by William Hendricks, who would be making a presentation summarizing past decisions and future directions.

"Hi, Bill," James said. "Will you have some new recommendations for us today? Are there any surprises from your research?"

"No surprises," Bill said. "But you will be pleased to know that our research has validated the decisions you have made recently. We are providing better service to the company than ever before, and we are doing it at less cost than we ever have before. I do have a few recommendations about how to fine-tune our infrastructure to provide even better service, though."

It was obvious that Bill was pretty enthused about the results of his research.

"Before we start, can you give me a brief idea about where the cost savings are coming from?" James asked.

"Sure," Bill said. "As you know, we decided to create a virtual private network using the Internet for all our supply chain and production needs. We moved all the computers to support this system into a colocation facility. We still own the servers, but we have signed a service agreement with the hosting company to maintain the operating system and network. This has allowed us to focus our efforts on the software itself and not distract valuable personnel with environment or connectivity issues. We also have not had to invest in additional buildings for a larger data center. Plus, the level of service is incredible. We have had almost 100 percent uptime since the switchover. The people in our plants really are pleased that they can check inventory levels and shipment dates from all their suppliers at any time."

"Wow. That is great news," James said. "And what are you going to recommend for our marketing and sales system?"

"Well, as you know, that is a Web-based system," Bill replied. "It doesn't have the security requirements that our production systems do, but it needs to be widely available. Our research has shown that we can deploy that system through a hosting company that provides cloud computing. We have the option of going with our colocation provider or using another company we have used in the past. I think this other company is going to give us some good price concessions and will still be able to provide excellent service."

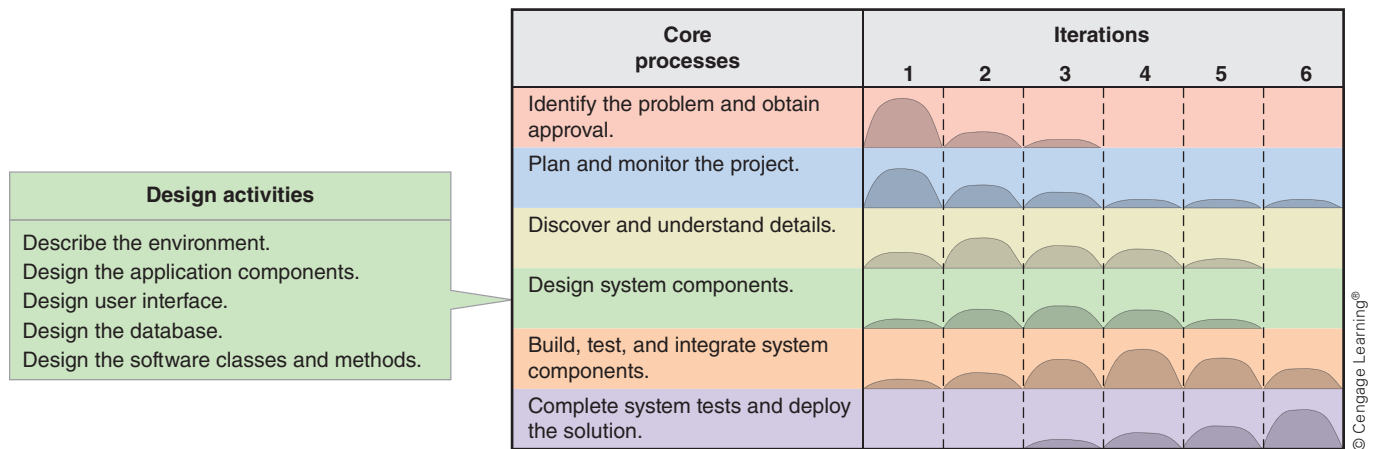
"That sounds great!" James said. "I'm interested in hearing about the details. I assume you have also laid out a migration plan to move the systems over?"

"Yes, I have done my homework on this one," Bill said. "I think you will be pleased with the results."

■ Overview

Requirements defined during analysis activities are usually stated in a technologically and architecturally neutral way. That is, they're stated in a way that enables designers to choose from a variety of technologies and architectures

FIGURE 7-1 Design activities



when deciding how best to satisfy the requirements. In contrast, system implementers develop and deploy software using specific technologies and within a specific architectural framework. Thus, an important part of design is choosing appropriate technologies, adapting those technologies and software to an organization's existing technology architecture, or modifying existing architecture to support new technologies and systems.

Technology and architecture are usually covered as part of an entire stand-alone course or in parts of several courses in most information system degree programs. Systems analysis is usually a separate course (or two). Because this book is targeted to students completing a systems analysis and design course, a complete coverage of technology and architecture isn't appropriate. Thus, this chapter serves two purposes:

1. Provide a review and summary of technology and architectural concepts most important in modern systems design
2. Describe the two design activities most concerned with technology and architecture: *Describe the environment* and *Design the application components* (see **Figure 7-1**)

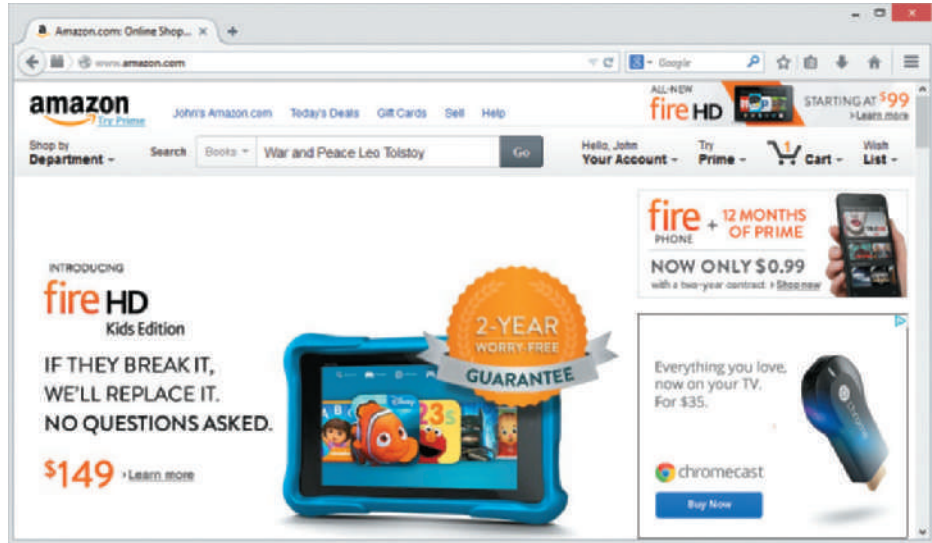
■ Anatomy of a Modern System

This section examines the technology and architectural features of a modern Web-based system—the Amazon.com shopping application. The discussion serves as a review of modern computer and software technology and an introduction to the architecture of Web-based systems. If you're already familiar with these topics, you can skip ahead to the "Architectural Concepts" section.

Amazon.com is the largest online retailer in the world. Most of you are familiar with this Web site (see **Figure 7-2**). Millions of users interact with this Web site every day to find products and make purchases (see **Figure 7-3**).

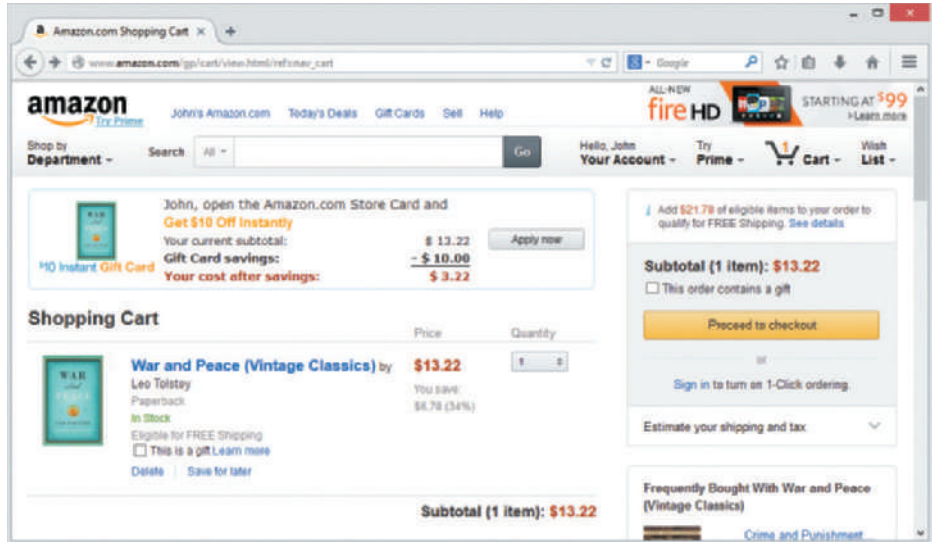
Figure 7-4 shows a simplified architectural diagram for the Amazon.com shopping application. Users interact with the application via a Web browser running on a personal computing device. Those devices normally connect to the Internet over wireless network connections and from there to one or more servers that host the Amazon.com shopping application. Other servers host additional software components to process payment and schedule shipments. We'll use this simple diagram as the starting point to examine the components and architecture of a typical modern information system.

FIGURE 7-2 Amazon.com main shopping page



Source: Amazon

FIGURE 7-3 Amazon.com product details



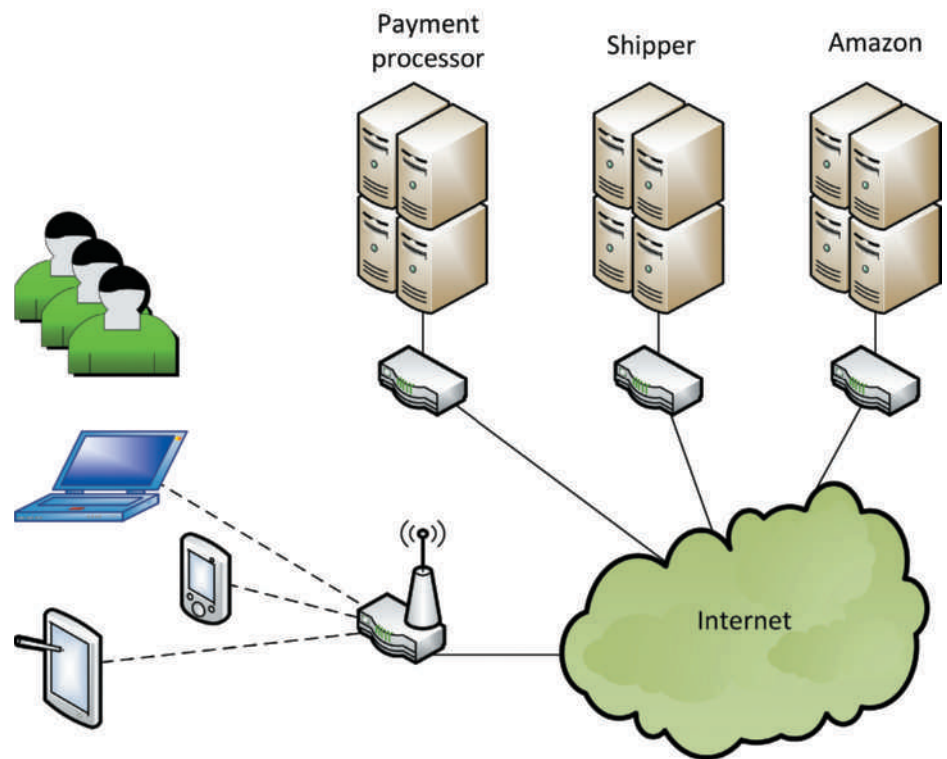
Source: Amazon

■ Computing Devices

server a computer or group of computers that manages shared resources such as file systems, databases, and Web sites, and enables users and other computers to access those resources over a network

The symbols along the top edge of Figure 7-4 represent servers. A **server** is a computer or group of computers that manages shared resources such as file systems, databases, and Web sites, and enables users and other computers to access those resources over a network. Examples of server roles include hosting a shopping application such as Amazon.com, managing e-mail for hundreds to millions of users, managing the flow of transactions through a stock exchange, and running the numerical simulations needed to forecast weather. Smaller organizations may own a handful of servers or may “rent” server capacity from larger organizations. Large computing-intensive organizations such as Amazon, Google, and Facebook operate server “farms” that fill entire buildings and are interconnected and replicated on a global scale (see **Figure 7-5**).

FIGURE 7-4 Amazon.com shopping application simplified architecture



The three icons in the lower-left corner of Figure 7-4 represent personal computing devices. Personal computing devices include desktop workstations, laptops, tablets, and cell phones, with a dizzying array of alternatives that often blur the categories. They have the same basic hardware components as servers—processors, memory, disk or flash storage, and input/output devices—tailored to satisfy the computing needs of one user at a time.

FIGURE 7-5 Server farm within a data center



© Eimantas Buzas/Shutterstock.com

■ Networks, the Internet, and the World Wide Web

The true power of modern computing lies not just in the ubiquity and power of individual computers, but in the ability to interconnect them. Social networking services such as Facebook, information resources such as Wikipedia and Google Search, and shopping applications like Amazon.com all rely on the interconnection of millions of computers, their users, and the resources that they manage.

A computer network is a collection of hardware, software, and transmission media that enables computing devices to communicate with one another and to share resources. Networks come in many sizes—ranging from small networks that span a single home or office to international networks that connect countries and continents.

Networks of all sizes interconnect with one another to form the Internet. The Internet is sometimes called the network of networks because it interconnects millions of private, public, business, academic, and government networks. The Internet acts a global highway for information exchange among computing devices and their users. The largest networks within the Internet are called **Internet backbone networks**. They provide the high-capacity, high transmission speeds, and reliability needed to carry billions of messages across regions, countries, and continents. Most are owned and operated by governments or large telecommunications companies.

At the other end of the network size scale are **local area networks (LANs)**, which typically span a single home, small office, or one floor of a building. Servers and personal computing devices connect wirelessly or via cables to LANs. LANs, in turn, connect to the Internet via larger networks spanning groups of buildings, cities, or regions.

The **World Wide Web**, sometimes called the **WWW** or simply the Web, is an interconnected set of resources accessed via the Internet. Though the terms *Internet* and *Web* are used interchangeably in everyday language, the distinction between them is important for information system designers. The Internet is the “highway” over which messages and information travel from network to network, server to personal computing device, and service provider to end user. The Web is the set of resources and services that can be accessed via that highway.

Web resources vary widely and include static documents (i.e., Web pages), images, videos, e-mail and other messages, and software-based services such as shopping sites and online games. Web resources are identified by a **Uniform Resource Locator (URL)** composed of three parts, as shown in **Figure 7-6**.

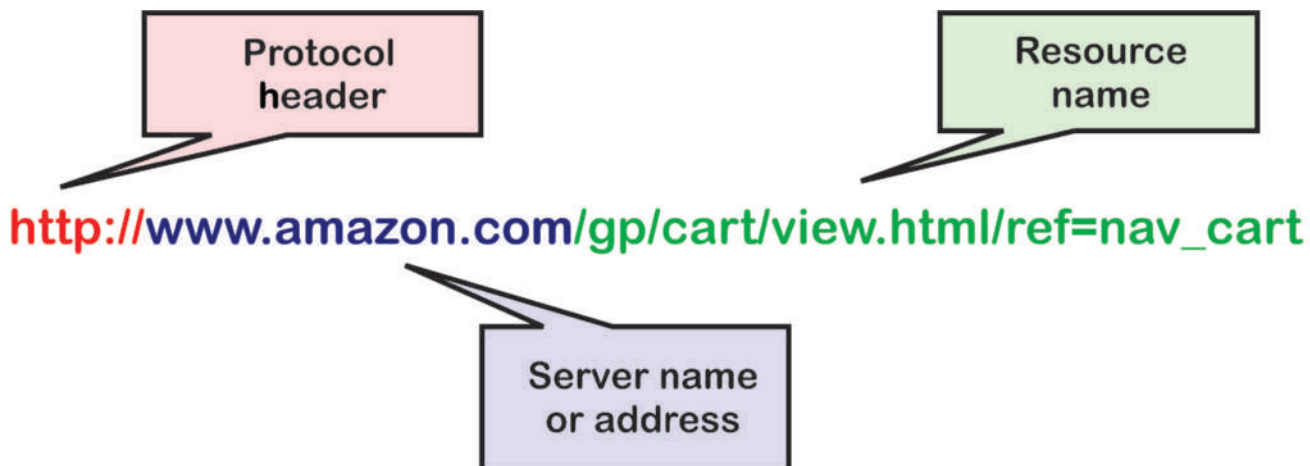
Internet backbone network a high-capacity and high-speed computer network that carries large amounts of Internet traffic across regions, countries, and continents

local area network (LAN) a small computer network typically spanning a single home, small office, or one floor of a building

World Wide Web (WWW) an interconnected set of resources accessed via the Internet

Uniform Resource Locator (URL) identifier of a Web resource containing a protocol header, server name or address, and resource name

FIGURE 7-6 Parts of a Uniform Resource Locator



hyperlink the URL of one Web resource is embedded within another Web resource

The URL of one Web resource can be embedded within another Web resource as a **hyperlink**. Documents, embedded hyperlinks, and the resources pointed to by those hyperlinks form a connection network that resembles a large spider web when diagrammed—thus, the word *web* within the term *World Wide Web*.

■ Software

The software components of the Amazon.com shopping application aren't visible in Figure 7-4 because they're embedded within the servers and personal computing devices. **Figure 7-7** shows the software components within blue callout boxes.

application software software that performs user- or business-specific tasks and is typically constructed as an app or Web-based application

app application software that is installed on the storage device of a computer or cell phone

system software software, such as operating systems and Web server software, that works behind the scenes to support application software and control or interact with hardware or software resources

Software components can be loosely grouped into two types. **Application software** includes software that performs user- or business-specific tasks, such as shopping, preparing customer purchase invoices, generating monthly financial statements, or enabling a user to play games against the computer or other users. Some application software is distributed as an **app**, which is installed once on a computer or cell phone's storage device. Other applications are constructed as Web-based applications that run within a Web browser and install little or no software on the user's device.

System software is software that works behind the scenes to support application software and to control or interact with hardware or software resources. Examples include operating systems (OSs), database management systems, Web browsers, and Web server software. OSs are an especially important class of business software because they're installed on every computing device and because they provide so many support functions used by application software. To a significant extent, the OS controls what a device can and can't do, thus enabling or limiting the application software that can be used on the device.

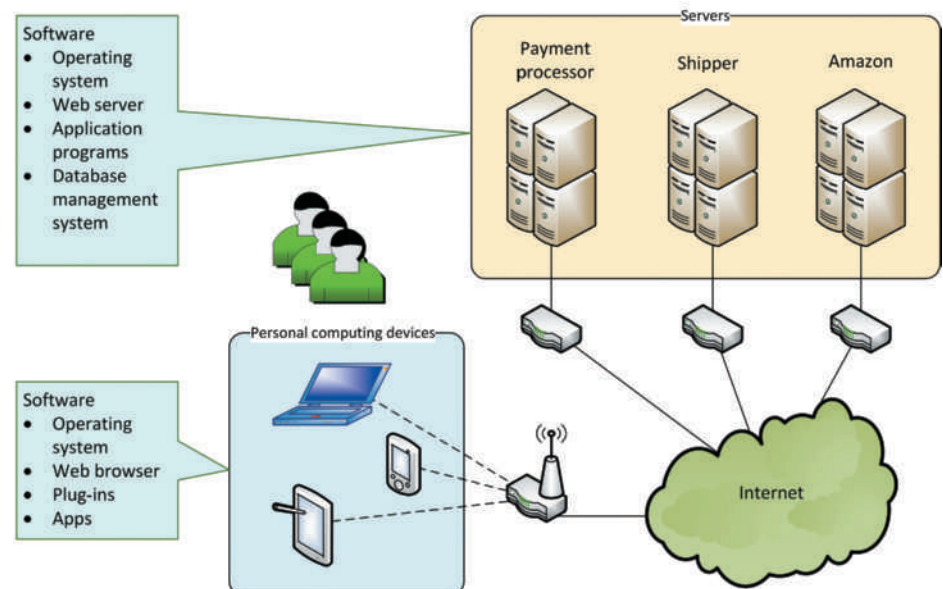
■ Web-Based Applications

Applications such as the Amazon shopping application are constructed as Web-based applications. Characteristics of **Web-based applications** include:

Web-based application application software that uses a Web browser as the user interface, has a URL for application access, uses a Web server and server-side software components, and uses Web standards for communication between Web browser and server

- Use of a Web browser as the primary user interface on personal computing devices
- User access to the Web application via a URL

FIGURE 7-7 Software components of the Amazon.com shopping application



- Server-side software components that execute on or are called from a Web server
- Use of Web standards for communication between Web browser and server

The reasons that Web applications are so common include ease of access and use of widely adopted Web standards. However, those benefits come at a cost. Web applications are a well-defined target for security breaches because URLs and Web standards are open and widely known. Also, application performance can suffer when networks are congested.

The impact of these challenges on the Amazon shopping application is significant. Amazon servers employ elaborate mechanisms to protect against threats such as denial-of-service attacks, bogus transactions, and interception of customer credit card and other sensitive information in transit. To ensure adequate performance for a global customer base, Amazon uses redundant and globally distributed servers connected to Internet backbone networks.

■ Embedded Software

A modern laptop computer, tablet computer, or cell phone comes preinstalled with a very complex OS, a Web browser, and a rich set of preinstalled apps. The OS has embedded software that provides functions for graphical display, touch screen and gesture-based input, voice recognition, speech generation, music and video playback, and access to the Internet and Web resources. Embedding these capabilities within the OS enables preinstalled or purchased apps and Web-based applications to reuse the software that implements these functions. That reuse provides important benefits, including a similar look and feel across applications and application development toolkits for software developers that automate many programming tasks and provide a direct interface to software already embedded in the device.

Embedded software components extend the functions of Web browsers and Web servers in ways that enhance Web-based applications and the user experience. Examples include the following:

- **Toolbars.** A set of links to Web resources or installed apps that extend the capabilities of a Web browser. For example, an Acrobat PDF toolbar provides Acrobat help links and one-click access to PDF document creation and management software.
- **Plug-ins.** Web browser plug-ins are often used to correctly display certain types of Web content. For example, Amazon provides plug-ins to enable users to preview 30-second music clips before purchasing and downloading digital music files.
- **Widgets.** An example of a Web browser widget is a time-and-temperature or stock market widget that is always displayed in one corner of the browser no matter what Web page is being viewed. An example of a widget installed within a Web page is a program that display statistics, such as user comments or visits to the page.

The combination of a computing device's OS, apps, Web browser, and embedded components creates a software environment within which an app or Web-based application operates. That environment provides a set of supporting functions for application software, but it also creates significant development, deployment, and support issues. One such issue is whether software developers deliver application software as an app or Web-based application. If they develop an app, they gain some advantages like improved performance, higher-quality user-interface features, and the ability to perform some functions while not connected to the Internet. But they also incur the costs of supporting multiple OSs. For example, if developers want their apps to be installable on most cell phones,

then they'll need to develop and support three different apps—one for iOS, one for Android, and another for Windows.

Developing application software that works on many device types—for example, desktops, laptops, and tablets (in addition to cell phones)—adds even more complexity. Fundamental differences in hardware and screen size forces developers to adapt application software in subtle and not-so-subtle ways. For instance, the keyboard and mouse/touch pad of a desktop or laptop computer may dictate one style of data input, while the small virtual keyboard and touch screen of a tablet or cell phone will dictate another approach. Screen size dictates significant differences between application software designed for a cell phone compared with software designed for a desktop computer and its large display.

The Amazon shopping application has an interesting approach to dealing with this complexity. Amazon doesn't provide installable apps for iOS, Android, and Google devices. Users with devices running those operating systems use a Web application, and Amazon invests considerable resources in developing server-side application software that adapts its user experience to the software environment of the user's device. However, Amazon has also released its own cell phone and a version of the Kindle e-reader tablet under the name Amazon Fire with a customized version of the Android OS. That OS includes embedded software and preinstalled apps that provide direct access to various Amazon services, including shopping. In essence, Amazon has developed a complete hardware and software environment that is fine-tuned for access to Amazon services. UPS and FedEx follow a similar strategy with the devices carried by delivery personnel to record, track, and process shipments.

In sum, the benefits of the rich embedded software architecture of most end-user computing devices are real—reuse of already-installed software, similar look and feel across multiple apps, and faster app development. But those benefits are coupled with significant problems of incompatibility and complexity for system designers and developers. Designers must decide which embedded software components will be included in their systems, and they must design their application software for that embedded software. Those choices determine the range of devices that can execute the application software and the costs of maintaining and upgrading it for years to come.

■ Protocols

protocol a set of languages, rules, and procedures that ensure accurate and efficient data exchange and coordination among hardware and software components

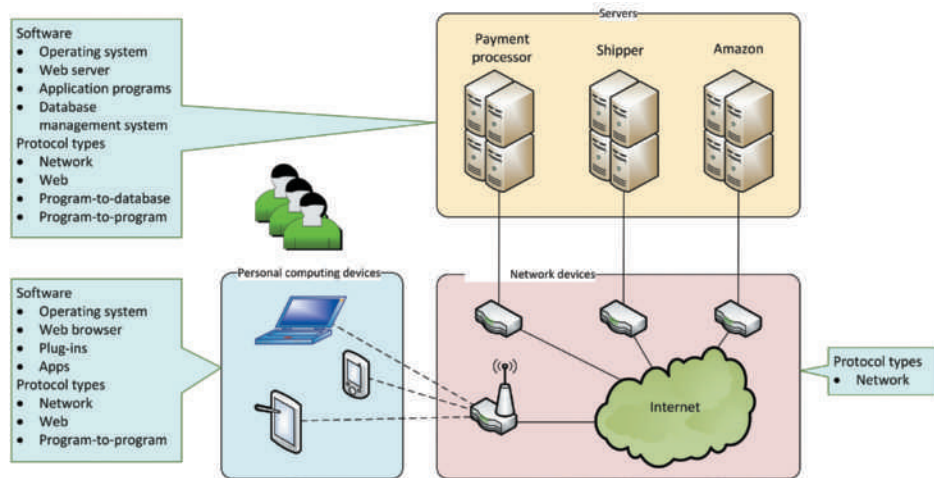
A **protocol** is a set of languages, rules, and procedures that ensure accurate and efficient data exchange and coordination among hardware and software components. Modern information systems rely on hundreds or thousands of protocols. Hardware, software, and network components abide by protocols because their owners and users value the results of accurate and efficient data exchange and coordination. Given the complex interconnected world of modern information systems, not abiding by protocols is unthinkable because no system could function without them. **Figure 7-8** shows types of protocols and their use within the Amazon.com shopping application.

■ Network Protocols

Network protocols enable accurate message transmission among the various computers and software components. They function as the “plumbing” that enables messages to find their way from sender to recipient, enable multiple devices to efficiently share wireless and wired connections, and handle tasks such as identifying and retransmitting lost messages. Network protocols are implemented within all computers and network hardware devices.

It is rare that an information system designer needs to be concerned with networking protocols other than their security implications. For purposes of

FIGURE 7-8 Software components and protocol types of the Amazon .com shopping application



information systems design, the primary challenge created by networking protocols lies in deciding which network addresses and URLs the software components will use to connect with one another. This is an important issue because many computers and networks employ software programs or hardware devices called firewalls. Firewalls examine incoming and outgoing packets and block those arriving from or sent to “dangerous” URLs or network address. Thus, an information system designer must ensure that network messages among software components won’t be blocked by a firewall at either end.

For increased security, the system designer may want to work with security and network specialists to develop a **virtual private network (VPN)** to isolate sensitive communications between servers or between an organization’s own employees and servers. A VPN incorporates technologies covered in Chapter 6, including Secure Sockets Layers, Transport Layer Security, and secure IP, to reliably identify senders and recipients, and encrypt network messages sent among them.

The Amazon.com shopping application doesn’t use secure network protocols for server-to-customer communication until the user needs to transmit sensitive data, such as when providing credit card information. All traffic among Amazon servers and those of its key partners, such as payment processors and shippers, is secured using virtual private networks.

■ Web Protocols

The World Wide Web is built on a small family of protocols for encoding Web documents and hyperlinks, requesting documents from Web servers, and responding to those requests. The most important protocols include the following:

- **URL.** As defined earlier, a URL defines a document or resource name, location, and encoding protocol.
- **Hypertext Markup Language (HTML).** HTML defines the structure and content of a Web page, including encoding methods for text, images, tables, and forms with the ability to control such aspects as text font and size, and the location of text and images on a displayed or printed page.
- **Extensible Markup Language (XML).** XML is an extension to HTML that enables groups of users to define additional encoding within Web documents. This encoding identifies the meaning of specific words,

virtual private network (VPN) secure communication over the Internet using technologies that reliably identify senders and recipients, and encrypt network messages sent among them

Hypertext Markup Language (HTML) a protocol that defines the structure and content of a Web page

Extensible Markup Language (XML) an HTML extension that enables the meaning of words, phrases, or numbers to be defined

Hypertext Transfer Protocol (HTTP) a protocol that defines the format and content of requests for Web documents and related data communication

phrases, or numbers. For example, in Figure 7-3 the text \$13.22 might include additional encoding information that identifies it as a product price stated in U.S. dollars. A price comparison service could use such information to compare prices for the same product across multiple shopping sites.

- **Hypertext Transfer Protocol (HTTP).** HTTP defines the format and content of requests for Web documents and resources, and includes some simple commands for other types of requests, such as uploading a file or page. For example, Web browsers normally request that a Web page be transmitted for display via the HTTP GET command. Data entered by a user in an HTML form is normally transmitted back to a Web server using the HTTP POST command.
- **Hypertext Transfer Protocol Secure (HTTPS).** HTTPS is an extension of HTTP that employs secure network protocols to reliably identify senders and recipients and encrypt network messages sent among them.

As with network protocols, Web protocols are nearly universal and thus of little direct concern to information system designers. The primary issue is which versions of Web protocols are supported by software on servers and end-user computing devices. For example, HTML5 is a recent update to earlier HTML protocol versions. It provides some new capabilities that Web application and Web service developers may want to incorporate into their systems. However, many users have older computing devices and software that don't support HTML5. Thus, a system developer must decide whether to

- Rely exclusively on an older HTML protocol version, sacrificing some functionality
- Rely only on HTML5 and thus not serve users with older or outdated devices
- Write multiple versions of software, one for each supported protocol version

Note that this sort of decision isn't unique to Web protocols. It occurs with other types of protocols and with support for specific operating systems.

A user's initial connection to the Amazon.com shopping application uses a URL that initiates the transfer of a Web page from an Amazon server to the user's Web browser. At that time, the Amazon Web server queries the user's Web browser to determine its type, what protocols it supports, and the underlying OS. That information is stored for the duration of the user's connection and used to adapt HTML documents to the specific characteristics and capabilities of the user's device. When sensitive data needs to be transmitted, the server initiates an HTTPS connection with the browser to verify its identity and encrypt/decrypt transmitted data. Data sharing among Amazon.com servers, including its key partners, uses XML encoding and employs secure network and Web protocols.

■ Architectural Concepts

In Chapter 2, we defined two key terms that we'll spend most of this chapter exploring:

- **Technology architecture.** The set of computing hardware, network hardware and topology, and system software employed by an organization
- **Application architecture.** The set of information systems (the software applications) the organization needs to support its strategic plan

The technology architecture defines the infrastructure that supports application software and the services it provides. Application software is deployed on a technology architecture by distributing application components to specific hardware devices and connecting them via networks and protocols.

Technology and application architecture are interdependent. Poor technology architecture provides a weak foundation for application software and can compromise its performance, reliability, and other characteristics. Good technology architecture enhances those characteristics and provides other benefits, such as minimal operating cost and flexible updating. But high-quality technology architecture can't make up for poor application architecture. Application software must be designed to ensure ease of construction, deployment, operation, and future updates.

Popular literature and culture usually refer to technology in abstract terms, and there is an underlying assumption that any useful technology will be adopted and exploited. But an organization must trade off the potential benefits of adopting and deploying useful technologies with the cost of supporting those technologies and the possibility of overextending its ability to manage and successfully exploit them. For example, should an organization immediately move its applications to “latest and greatest” technologies, or should it wait until it becomes clearer which of those technologies are useful and widely adopted? Once it decides to adopt a technology, which “flavor(s)” should it choose? What specific hardware, software, and protocols should it deploy and support?

These are important questions because no organization has sufficient resources to constantly reimplement existing information systems using all possible new and updated technologies. An organization must choose technologies that match its application needs and support those technologies over a period of years or decades. The technology architecture must evolve slowly over time to enable new functions, new ways of working, and new ways of interacting with customers and strategic partners.

■ Software as a Service

A service is something that we purchase rather than making or doing it ourselves. For example, we consider our household utilities such as water, electricity, and trash pickup to be services. We don't have to have our own power generator to get electricity. We just buy power as we use it. Another example is service on our vehicles. When something breaks on our cars, we pay an auto mechanic to fix it. We don't usually operate our own repair shop. We purchase repair services from someone who does.

Software as a service (SaaS) follows that same basic idea. If an organization requires some services—for example, bookkeeping and accounting functions—it could build or buy an accounting software system. Alternatively, it could find a firm that provides accounting services and buy only the accounting services it needs. As with any other utility service, it would purchase and pay only for those services it requires. It doesn't have to purchase—or install or maintain—the software system.

Many applications employ the SaaS model. Examples include the following:

- Social networking services such as Facebook, Tumblr, and Pinterest
- Application software suites such as Google Apps and Adobe Creative Suite
- Specialized applications such as the Amazon.com shopping application and the UPS.com shipping application

Two common features shared by most applications that employ the SaaS model include the following:

- Little or no application software is installed on the user's device.
- User data is stored on servers, though copies may be stored on the user's device for improved performance.

software as a service (SaaS) a software delivery model similar to a utility, in which application software is accessed via the Internet without locally installed programs

■ Web Services

A Web service is a software service accessed over the Internet using Web protocols. Although the term can be used to refer to applications such as those listed above, it more commonly refers to smaller and more focused applications, such as transmitting shipping information from seller to shipper, or single functions, such as looking up the zip code that matches a shipping address. In those contexts, a **Web service** is a software function or related set of functions that can be executed via Web standards. In essence, a Web service is a software function, subroutine, method, or program that meets the following criteria:

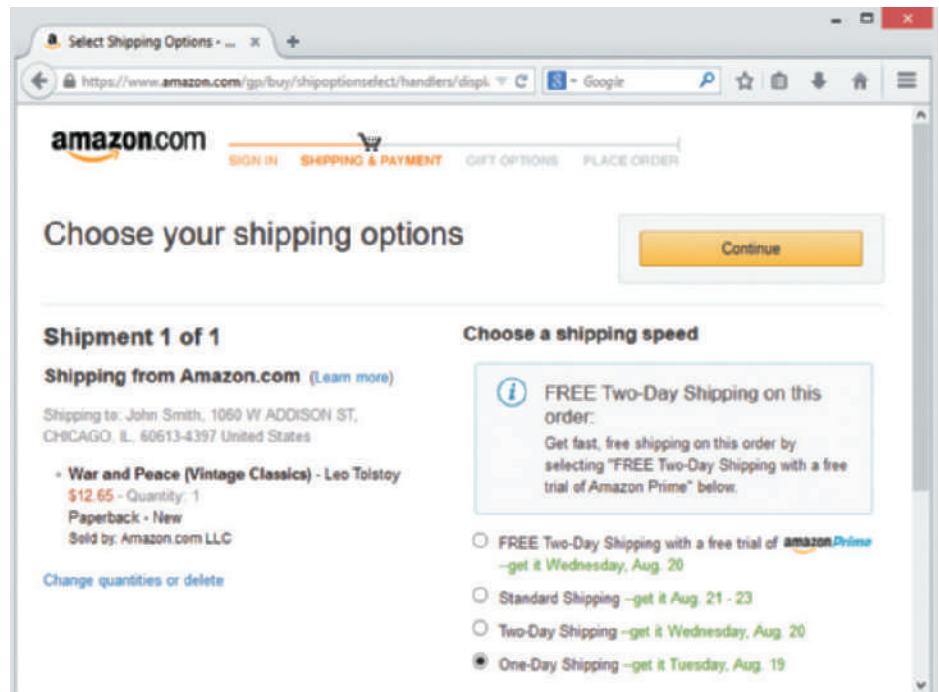
Web service software function or related set of functions that can be executed via Web standards

- Is “called” from one application via a URL or other Web protocol
- Accepts input data embedded within the URL or via another protocol
- Executes on the Web service owner’s servers
- Returns processing results encoded within a Web page or document

Web services enable software functions developed by organizations to be embedded within the information system of another organization. For example, consider the shipping options for an Amazon.com purchase, as shown in **Figure 7-9**. If the user selects the one-day shipping option, how will the Amazon Web application determine what shipping charge to apply to the order? The Amazon Web application knows the address of the warehouse that holds the items to be shipped and the customer’s address. It also knows the approximate dimensions of a box that will hold the purchased item, and it can calculate the weight based on the weight of the enclosed items, box, and packing materials. What the Amazon.com Web application doesn’t know and can’t calculate itself is how much a shipper such as FedEx or UPS will charge to deliver the box.

Shipping companies provide Web services that enable companies like Amazon.com to compute shipping charges. Amazon’s Web application passes shipment data to the Web service, and the shipper’s software computes the charge and then passes it back to the Amazon.com Web application. In essence, the Amazon.com Web application executes the shipper’s Web service as a subroutine over the Web. In a similar manner, the Web application can interact

FIGURE 7-9 Shipping options for an Amazon.com purchase



Source: Amazon

with the Visa or MasterCard Web service to approve and process a charge to the customer's credit card. For items that are out of stock, the Web application can interact with publisher Web services to locate available sale items and estimate the time to ship them to an Amazon warehouse or direct ship then from the publisher warehouse to a customer.

The implications of Web services for system design are significant. Summarized at the strategic level, information system developers must do the following:

- Scan the range of available Web services and decide which to incorporate into their software. To the extent they include other Web services, they expand the functions of their own software with minimal related development cost.
- Decide which (if any) functions of their own software should be implemented as Web services and made available to other systems. To the extent they do so, they increase the potential base of software users, but also commit to making those services available in a reliable and secure fashion over a long time period.

At a more detailed level, the decision to use or provide Web services leads to many other questions about system design, development, and deployment, including the following:

- What protocols will be used to accept Web service calls and to pass data in both directions?
- How will data passed to/from Web services be secured?
- Even if no Web services are made available to external users, should some portions of the system be structured as Web services to facilitate access and use by other internal systems?
- What performance and availability guarantees will be provided to Web service users?
- What server, network, and other resources are needed to satisfy those guarantees?

We'll revisit some of these questions in the examples later in this chapter.

■ Distributed Architectures

Distribution of system components across machines, server farms, and oceans is an inescapable facet of modern system architecture. Consider again the Amazon shopping application shown in Figure 7-8. Participating organizations (e.g., Amazon, Visa, MasterCard, UPS, and FedEx) each operate their own server farms in the locations of their choice. Further, to serve a global market and to provide higher performance and fault tolerance, each organization distributes its servers and software across dozens of locations throughout the world. Thus, a shopping session for a customer in California might be processed in an Amazon server farm in Canada. The Canadian servers might interact with UPS servers in Japan and Visa servers in Germany.

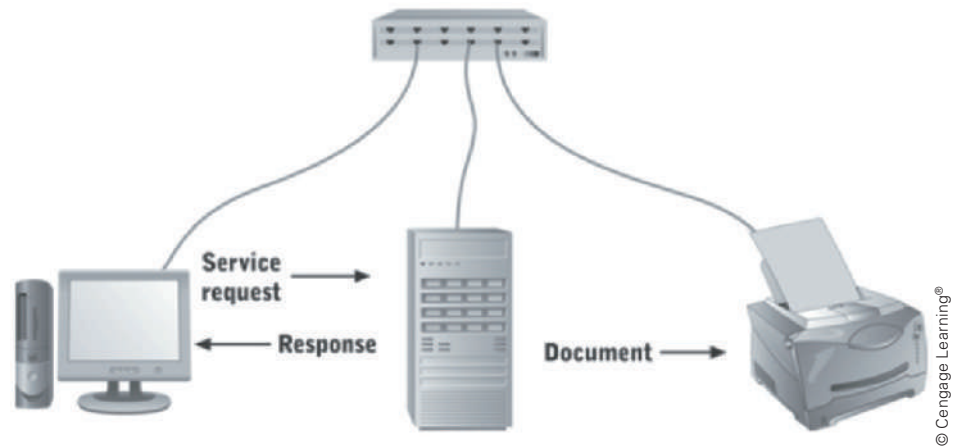
■ Client/Server Architecture

Client/server architecture is a method of organizing software to provide and access distributed information and computing resources. It divides software into two classes: client and server. A server manages system resources and provides access to these resources through a well-defined communication interface. A client uses the communication interface to request resources, and the server responds to these requests.

The client/server architectural model can be applied in many ways. A simple example is how desktop computers access a shared printer on a LAN, as shown in **Figure 7-10**. An application program on a desktop computer sends a

client/server architecture a software design and deployment method that divides software into components that manage resources and components that use those resources

FIGURE 7-10 Network printing services implemented with client/server architecture



document to a server that dispatches it to a management process for the specified printer. The server acknowledges the client request and notifies the client when the document is sent to the printer.

Note that in the context of client/server architecture, the term *server* doesn't refer to computer hardware. Rather it refers to software that might be embedded within an operating system, be implemented as a separate system software component such as a database management system, or provided as a service accessible via Web protocols. A server within client/server architecture can be hosted within a hardware device, such as a smart printer, or one or more computer systems called servers. Server computer systems might host a single service or many different services. Also, a single service might be replicated across many different computing systems or server farms to make its services available to many geographically dispersed clients.

In Figure 7-8, the customer's laptop computer acts as a client while talking to the server components of the Amazon.com shopping application. In turn, the Amazon.com shopping application acts as a client to shipper and payment-processing Web services. Thus, it is possible (and common) for a single software component or system to be both client and server.

■ Three-Layer Architecture

three-layer architecture a client/server architecture that divides an application into view layer, business logic layer, and data layer

view layer the part of a three-layer architecture that contains the user interface

business logic layer also known as the domain layer, the part of a three-layer architecture that contains the programs that implement the business rules and processes

data layer the part of a three-layer architecture that interacts with the data

Three-layer architecture is a variant of client/server architecture that is used for all types of systems, from internally deployed desktop applications to globally distributed Web-based applications. Three-layer architecture divides the application software into three layers that interact, as shown in Figure 7-11:

- The user interface or **view layer**, which accepts user input and formats and displays processing results
- The **business logic layer** or *domain layer*, which implements the rules and procedures of business processing
- The **data layer**, which manages stored data, usually in one or more databases

FIGURE 7-11 Three-layer architecture

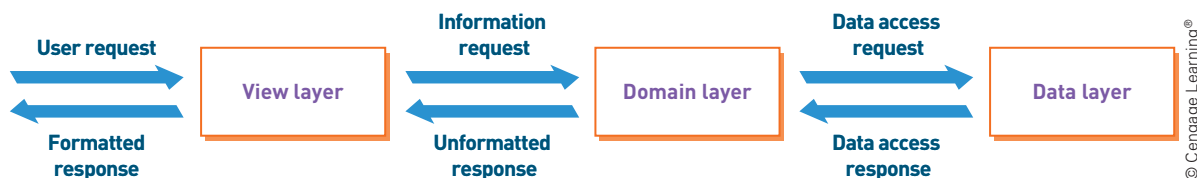
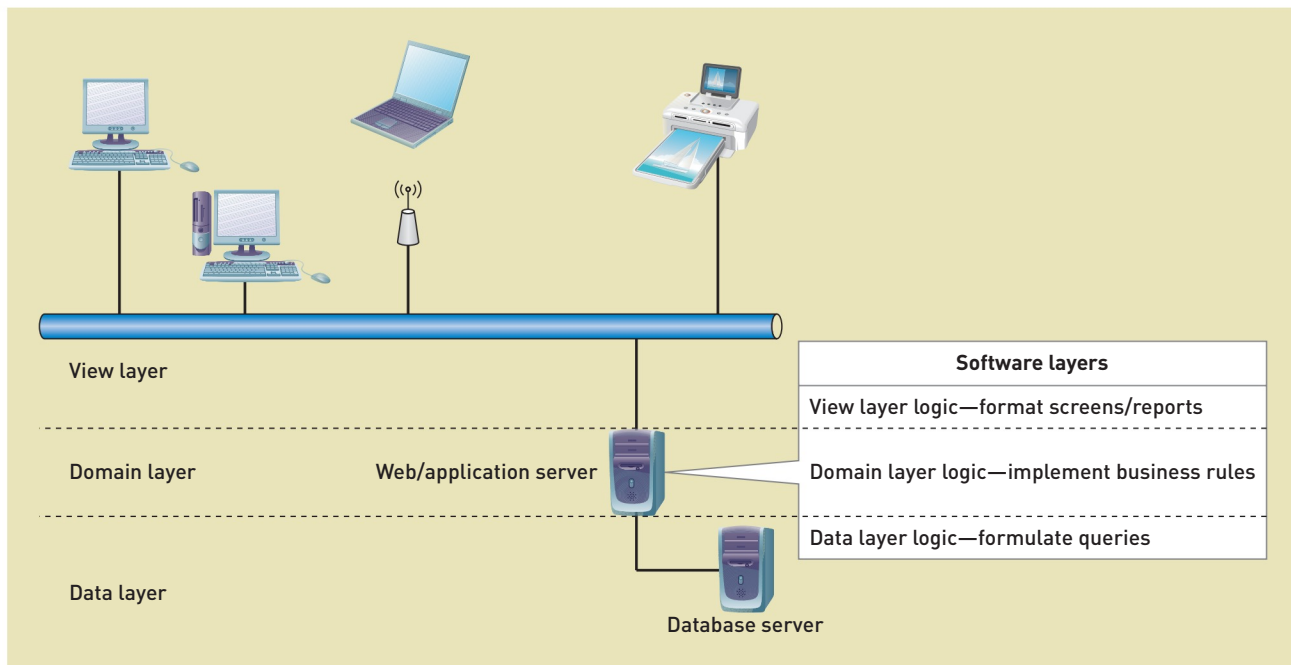


FIGURE 7-12 Internal deployment with three-layer architecture



One of the advantages of a client/server architecture is that it easily supports—in fact, encourages—software to be developed by using three-layer architecture. **Figure 7-12** illustrates an internally deployed system with a three-layer architecture, and it shows how the three layers might be configured across three separate computing platforms: desktop and laptop clients, a Web/application server, and a database server.

The view layer resides on all the client computers. The HTML is interpreted and displayed by the Web browser on the client computers. The view layer software components that format the HTML are on the application server. The data layer consists of the database server and any application programs on the application server that are necessary to access the data. The business logic layer resides on the application server computer and includes all the logic to process the business rules.

A major benefit of three-layer architecture is its inherent flexibility. Interactions among the layers are always requests or responses that follow a Web or other protocol, which make the layers relatively independent of one another. It doesn't matter where other layers are implemented or on what type of computer or operating system they execute. The only interlayer dependencies are a common protocol for requests and responses and a reliable network with sufficient communication capacity.

Multiple layers can execute on the same computer, or each layer can operate on a separate computer. Complex layers can be split across two or more computers. System capacity can be increased by splitting layer functions across computers or by load sharing across redundant computers. In the event of a malfunction, redundancy improves system reliability because the server load can be shifted from one computer to another. In sum, three-layer architecture provides the flexibility needed by modern organizations to deploy and redeploy information-processing resources in response to rapidly changing conditions.

■ Interoperability

interoperability the ability of a component or system to interact with other components or systems

It's natural to feel a bit overwhelmed at this point in the chapter. We've described hardware, software, protocols, and multiple architectural concepts. You've encountered many new terms and acronyms and have developed some new perspectives on things that you thought you already understood. You're probably searching for a framework into which you place all of the new and updated information. You're also wondering: What does it all have to do with systems design?

Interoperability is a perspective through which you can examine all of the details covered thus far. **Interoperability** is the ability (or lack thereof) of a component or system to interact with other components or systems. The myriad hardware and software components that make up a modern information system must work together. To ensure interoperability, system designers must do several things, including the following:

- Understand and describe the current environment in which the system will operate. That environment includes existing hardware and software components, networks, and the protocols and APIs already in use.
- Look for existing software components and services that can provide needed functions for the new system. Those functions must be interoperable with the existing environment and with components that will be built.
- Build components that can't be purchased as software modules or used as a service. As with acquired components, these must be interoperable with all other system components and with existing environments.
- Structure and assemble the components in a way that is feasible to build, test, deploy, and operate over the long term.

That sounds like a tough and complex job because it is. Successful system designers are knowledgeable and experienced. They acquire a broad and deep knowledge of all of the topics discussed thus far in this chapter. They learn to apply that knowledge by working as developers and by gradually completing design tasks for larger and more complex subsystems and systems. They constantly update their knowledge to keep up with rapidly changing technology. And they work in teams because no one person can hold and successfully apply all of the knowledge needed to design and deploy a large-scale modern information system.

■ Architectural Diagrams

Architectural diagrams are commonly used to visually describe an information system's environment, components, and deployment. As with other diagrams used in systems design, architectural diagrams summarize complex details in a way that's easy for people to understand. The following sections briefly look at three widely used architectural diagrams: location, network, and deployment diagrams.

■ Location Diagrams

Location diagrams are commonly used to show the geographic placement of various system components, including hardware, buildings, and users. Location diagrams are developed early in the design process to describe a system's environment. A simple example from the RMO case will suffice to show how location diagrams are constructed.

RMO's main offices consist of the corporate headquarters as well as a large retail store, a manufacturing plant, and a large distribution warehouse in Park City, Utah. Park City is where the company got its start and where it opened its first retail store. Salt Lake City is, in many ways, the hub for RMO's daily

FIGURE 7-13 Location diagram showing RMO's warehouses, manufacturing plants, and retail stores



operations. The primary data center is located in a separate building in Park City. There are two distribution centers: one in Portland, Oregon, and the other in Albuquerque, New Mexico. Additional manufacturing is done in Seattle, Washington. **Figure 7-13** is a location diagram showing retail, manufacturing, and distribution locations.

■ Network Diagrams

network diagram a model that shows how locations and hardware components are interconnected with network devices and wiring

A **network diagram** shows how locations and hardware components are interconnected with network devices and wiring. There are many different kinds of network diagrams—each emphasizing different aspects of the network, connected hardware resources, and users.

Figures 7-4, 7-7, and 7-8 are all network diagrams. **Figure 7-14** shows another network diagram that focuses much more specifically on network connections and network hardware. A diagram of this type is typically used to describe network connections in a building or within a server room.

■ Deployment Diagrams

A deployment diagram describes how software components are distributed across hardware and system software components. **Figure 7-15** shows software subsystems and database components distributed across two servers.

FIGURE 7-14 Network diagram showing building or server room details

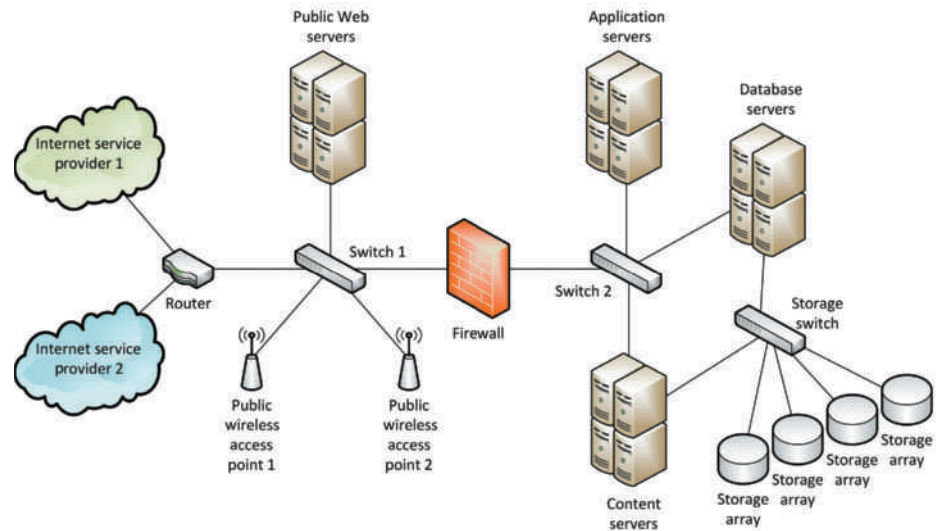
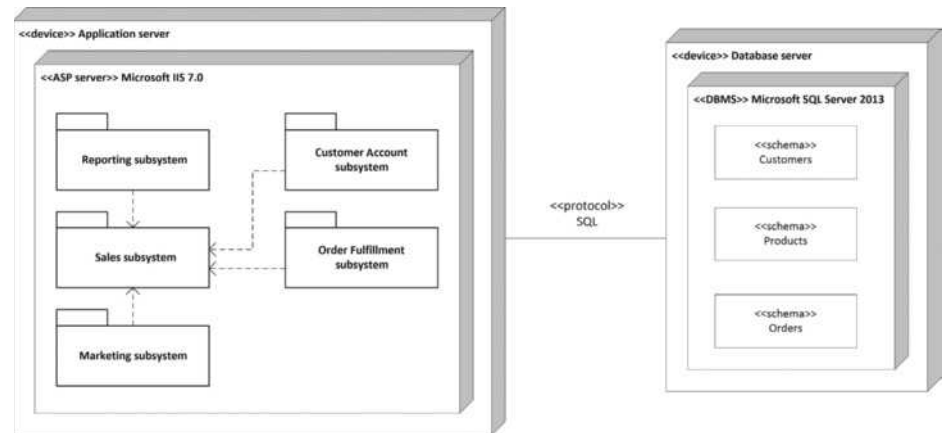


FIGURE 7-15 Deployment diagram for RMO subsystems



On the left, an application server hosts Microsoft Internet Information Server, which in turn hosts multiple CSMS subsystems. The subsystems and their dependencies are represented as embedded package diagrams. On the right, a database server hosts Microsoft SQL Server 2013 as the database management system (DBMS). In turn, the DBMS hosts three database schemas that provide permanent storage of customer, order, and product data. Software components in the application server communicate with the DBMS using SQL as the primary protocol and database access language.

■ Describing the Environment

Returning for a moment to Figure 7-1, notice that describing the environment is the first design activity. As discussed in Chapter 6, that description includes two key elements:

- External systems
- Technology architecture

A large-scale information system such as RMO's CSMS will typically interact with a few dozen external systems, including other systems owned by RMO (e.g., financial reporting and supply chain management systems) and systems owned or operated by other organizations (e.g., payment processors

and shippers). Furthermore, a new system isn't developed in a technology vacuum. Rather, the owning organization has many existing systems and a technology architecture that supports them. Thus, any new system must integrate with that technology architecture. If the new system requires technology architecture changes, they must be carefully considered and coordinated with the existing architecture and systems.

You've already reviewed many key aspects of external systems and technology architecture. You've also learned about three different types of diagrams that can be used to summarize environment information. The rest of this section first explores some of the key questions that must be answered when describing the environment. It then looks at a specific RMO example to show you how those questions are applied in practice. Finally, it discusses some specific issues of data integration that must be addressed when interconnecting multiple application components and systems.

■ Key Questions

A useful way of starting to describe the environment is to pose a series of questions. The answers to those questions provide a pool of information to be summarized in diagrams and supporting text. The following questions are a typical starting point:

1. What are the key features of the existing or proposed technology environment that will support or constrain the system?
 - a. What operating systems will be used?
 - b. What other system software (e.g., Web server, database management, and intrusion detection software) will be used?
 - c. In what ways are network messages filtered or otherwise secured? Are any changes required to support interactions with external systems or user-interface devices?
 - d. What APIs and development tools are compatible with the existing technology environment?
2. With what external systems and databases will the system under development interact? For each system or database, answer the following questions:
 - a. What is the timing and frequency of each interaction?
 - b. What is the data content of inputs to and outputs from the system?
 - c. What protocols will format and encode data flowing to or from the external system?
 - d. What are the security requirements of each inflow and outflow?
 - e. What security methods and protocols will be used to satisfy the security requirements?
3. What devices will be used for automated inputs and outputs?
 - a. What protocols will format and encode data flowing to or from the devices?
 - b. What are the security requirements of each inflow and outflow?
 - c. What security methods and protocols will be used to satisfy the security requirements?
 - d. What APIs and development tools are compatible with the existing technology environment and required automated inputs and outputs?
4. What user-interface technology will be used?
 - a. Where will users be located?
 - b. What hardware device(s) will users use?
 - c. What operating systems will run on "smart" user-interface devices?

- d. On what other user device software will the system rely (e.g., browsers, plug-ins, and software utilities embedded in the device)?
- e. What protocols will format and encode data flowing to or from user devices?
- f. What are the security requirements of each inflow and outflow?
- g. What security methods and protocols will be used to satisfy the security requirements?
- h. What APIs and development tools are compatible with the existing technology environment and required user interfaces?

Once the answers to these questions have been recorded, the designer can then begin developing architectural diagrams to summarize system interfaces and technology architecture. However, some details of those diagrams must be revisited when designers complete the next design activity: design the application components.

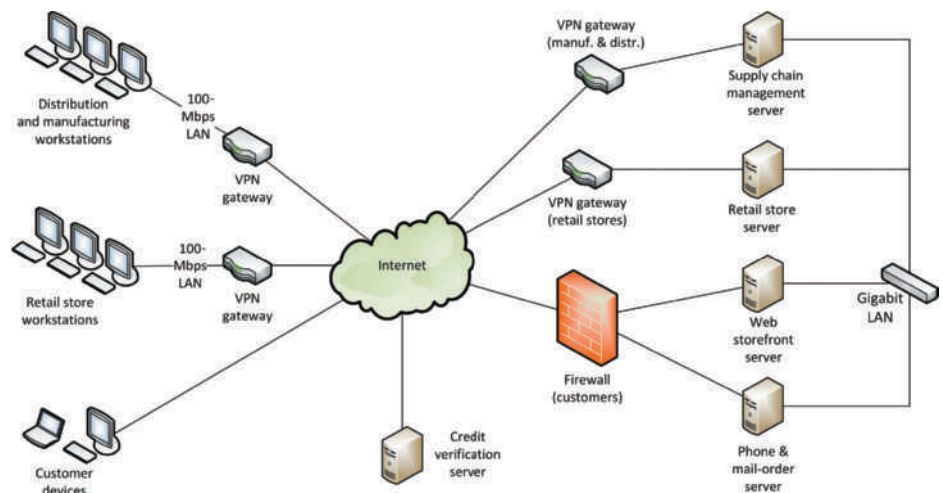
■ RMO Environment Description

Figure 7-16 shows a network diagram describing RMO's current technology architecture. Along with the data center in Park City, RMO has internal networks in every office, warehouse, manufacturing plant, and retail store. In addition, the distribution centers and manufacturing plants are all connected to the central data center and to the corporate offices in Park City by a VPN. Retail outlets are connected by a separate VPN to the central site. Separate servers host each primary system and a Gigabit LAN connects all servers and users in the data center and corporate offices.

Chapter 2 described several planned updates to the current system to develop the CSMS. This section focuses on a handful of updates to demonstrate how the environment is affected:

- Support for mobile end-user computing devices, including dedicated apps for tablets and smartphones
- Updated support for Web technologies, including user content adapted to device characteristics and structuring of key system components and functions as Web services
- Direct interface to social networking sites to enable opinion sharing about RMO products
- Consideration for the use of external service providers to host all or part of the CSMS

FIGURE 7-16 RMO's current technology architecture



The following subsections briefly address the impact of these updates as answers to the key questions posed above.

■ Mobile Devices and Apps

The current technology architecture connects users to the system using desktop or laptop computers. Expanding the range of user devices to include tablets and smartphones will require updates to the supporting system software, APIs, and development tools. The current system relies on servers running the Windows operating system with Internet Information Server as the Web server. That infrastructure is sufficient to support apps in all of the current flavors (iOS, Android, and Windows) though additional plug-ins will have to be installed and configured in each device to support each app type.

■ Web Technologies and Adapted Content

Because some users won't install an app, the CSMS must also support a browser-based user interface with support for multiple screen sizes, Web browsers, and plug-ins. That will require more complex user-interface coding than exists in the current system, which simply serves static Web pages and forms. The updated user-interface coding will need to query the user's device and browser and adjust the content of the Web pages transmitted to match the device characteristics. As with the apps, this may require supporting more modern Web protocols such as HTML5 and updated APIs and development tools.

Structuring newly deployed applications as Web services will require changing the way that RMO configures and manages servers. Note that each server in Figure 7-16 is dedicated to a specific system. Modern technology architecture that supports Web services generally organizes servers by type of service (e.g., Web, application, database, or video content) with each server hosting services or components from multiple information systems. In addition, groups of servers are generally deployed for each service type to provide service continuity in case one machine fails.

■ Social Networking

Some of the impacts of incorporating social networking into the system mirror those described earlier for apps and Web pages that adapt to the user's device. Social networking services provide the ability to interface with external systems in two ways:

- A Web services interface
- An API and toolkit that enable developers to create customized functions and embed them within the social networking site and interface

Additional research is required to determine whether one or both of these methods are needed to implement the social networking capabilities of the new CSMS. That research must be repeated for each social networking service (e.g., Facebook, Twitter, Pinterest, and Google+) with which the CSMS will interact. Of course, the choice of interaction method, the Web services protocols, and the API and development toolkit will vary among the social networking services.

■ Security Implications

Supporting apps, multiple browsers with plug-ins, and interfaces to social networking sites will probably require security updates to the current technology architecture. Apps will need to be digitally signed prior to distribution via the app stores for each device's operating system. In addition, some browsers and plug-ins require transmission of digitally signed code fragments that are encrypted with the organization's public key. Similarly, social

networking services implement stringent security protocols to protect themselves against malicious plug-ins and intrusions via Web services. Designers of the updated CSMS will have to work closely with system and network administrators to determine what updates to security configuration will be required.

■ Updated RMO Technology Architecture

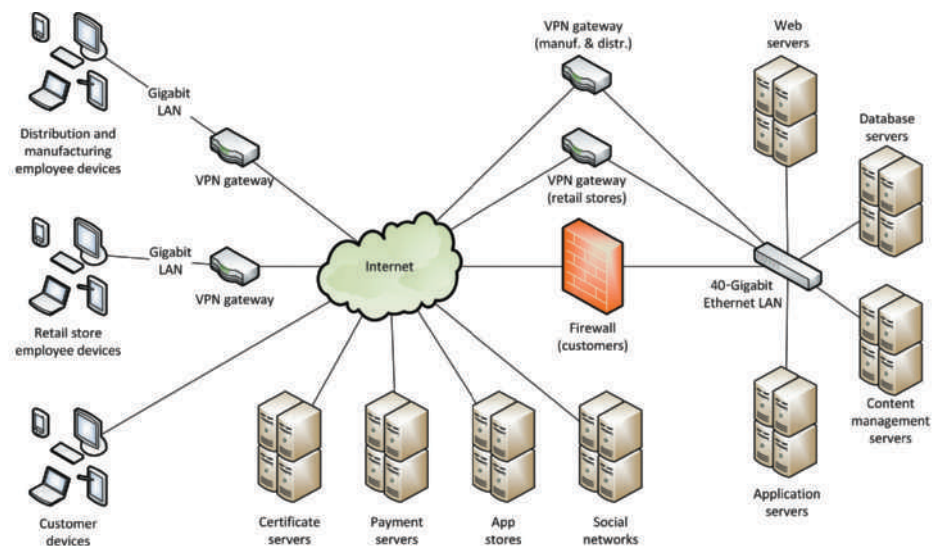
Figure 7-17 updates Figure 7-16 to explicitly incorporate support for mobile devices, apps, and social networking. Note the increased complexity of the diagram, including additional user-interface devices, upgraded network speeds, an increase in the number of servers, grouping servers by service type, the addition of content management servers for images and videos, and additional external service providers. The diagram doesn't capture additional complexity associated with updated security configuration and more complex software hosted within RMO's application servers. Increased network bandwidth is needed throughout the architecture due to use of Web services, the increased complexity of user interfaces, and increased use of graphics and video. Note that transition from the old technology architecture to the new may take months or years due to the need to redeploy and update legacy systems not replaced by the CSMS.

■ External Hosting

The right side of Figure 7-17 summarizes a lot of complexity and cost. Updates to the data center will be expensive, as will operating, managing, and maintaining it over time. Further, locating all key software and hardware assets in a single data center is risky. RMO risks customer dissatisfaction and loss of sales revenue if the data center were to be offline for any reason.

External hosting of all or part of the CSMS is an option that can reduce risk, improve performance, and possibly reduce cost. Hosting the application with a national or global company, such as Google or Amazon, would enable RMO to take advantage of an existing and highly distributed computing infrastructure. Key components could be replicated at multiple locations to improve performance and to provide fault tolerance. Costs might be lower due to improved economies of scale. Even if costs were the same or slightly higher, they might be justified by improved performance and fault tolerance.

FIGURE 7-17 RMO's updated technology architecture for the CSMS



■ Designing Application Components

Chapter 6 defined an application component as a well-defined unit of software that performs one or more specific tasks. That definition masked some important details, including variations in component size ranging from single subroutines or methods to entire subsystems; variations in programming language, protocols, and supporting system software; and the ability to build, buy, or freely access components as Web services of entire SaaS systems. This section concentrates on defining the functions and boundaries of larger application components, including Web services and subsystems. Design of smaller application components, such as classes and packages, is covered in later chapters.

■ Application Component Boundaries

Systems analysis activities collect and document various aspects of systems requirements that are needed to design application components. A key question to be answered when designing application components is which components will perform which functions? Answering this question can be approached from a top-down or bottom-up perspective. With a top-down approach, the designer thinks of the entire system as a single component performing all of the functions described during analysis activities. The designer then breaks this large component into smaller components in a generic process called factoring. With the bottom-up approach, the designer considers each function separately and looks for similarities as a basis for grouping software that implements the functions into larger application components.

With either approach, the designer looks for similarities among system functions to guide factoring or grouping. But how does a designer determine or measure similarity among system functions? When answering that question, it helps to focus on specific analysis activity descriptions like events and use cases. Events and use cases incorporate attributes that can be used to measure similarity, including the following:

- **Actors.** Each use case identifies one or more specific actors. Software for use cases that interact with the same actors is a candidate for grouping into a single application component.
- **Shared data.** Use cases that interact with the same domain class(es) are candidates for grouping into a single application component.
- **Events.** Use cases that are triggered by the same external, temporal, or state event are candidates for grouping into a single application component.

■ RMO CSMS Application Architecture

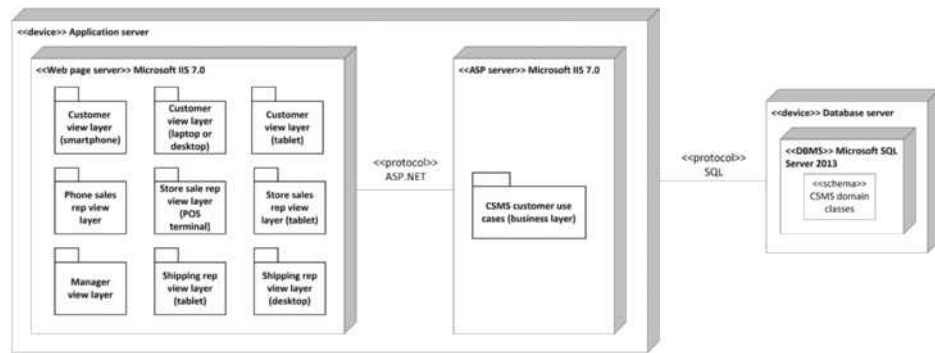
Let's illustrate how use case similarities can be used to define the boundaries of application components for a portion of RMO's CSMS. **Figure 7-18** combines information about use cases, domain classes, and events as defined in earlier chapters and shows all use cases in which Customer is an actor or user. The far-right column shows prospective groupings of those use cases, which are discussed further below.

Groups A–D include all use cases related to sale or return activities that can be performed online by the customer (Groups C and D) or with the assistance of a phone (Groups A and C) or store sales representative (Groups B and C). Group D includes use cases that are unique to the customer's online experience, including the online shopping cart. Group E cases cover various nonshopping aspects of the customer experience, such as product ratings and friend links. In essence, Groups C–E describe the customer's entire online experience while interacting with the CSMS.

FIGURE 7-18 CSMS use cases for the Customer user/actor

Use case	User/actor	Domain class(es)	Event(s)	Group
Create phone sale	Phone sales representative	ProductItem, InventoryItem, SaleItem, Sale, SaleTrans	Customer request while shopping by phone	A
Create store sale	Store sales representative	ProductItem, InventoryItem, SaleItem, Sale, SaleTrans	Customer request while shopping in store	B
Create/update customer account	Customer, phone or store sales representative	Customer, Account, Address	Customer request or sale to a new customer	C
Look up order status	Shipping, customer, management, phone or store sales representative	ProductItem, InventoryItem, SaleItem, Sale, SaleTrans, Shipment, ReturnItem	Customer, representative, shipping, or management request	
Track shipment	Shipping, customer, management, phone or store sales representative	Shipment, Shipper, SaleItem	Customer, representative, shipping, or management request	
Create item return	Customer, phone or store sales representative	SaleItem, ReturnItem	Customer requests return	
Search for item	Customer, phone or store sales representative	ProductItem	Customer request while shopping online, by phone, or in store	
View product comments and ratings	Customer, phone or store sales representative	ProductItem, ProductComment	Customer request while shopping online, by phone, or in store	
View accessory combinations	Customer, phone or store sales representative	ProductItem, AccessoryPackage	Customer request while shopping online, by phone, or in store	
Fill shopping cart	Customer	ProductItem, InventoryItem, CartItem, OnlineCart	Customer request, usually after sale completed	D
Empty shopping cart	Customer	ProductItem, InventoryItem, CartItem, OnlineCart	Customer request while shopping online	
Check out shopping cart	Customer	ProductItem, InventoryItem, CartItem, OnlineCart, SaleItem, Sale, SaleTrans	Customer request while shopping online	
Fill reserve cart	Customer	ProductItem, InventoryItem, CartItem, OnlineCart	Customer request while shopping online	
Empty reserve cart	Customer	ProductItem, InventoryItem, CartItem, OnlineCart	Customer request while shopping online	
Convert reserve cart	Customer	ProductItem, InventoryItem, CartItem, OnlineCart	Customer request while shopping online	
Rate and comment on product	Customer	Customer, ProductComment, ProductItem	Customer request, usually after sale completed	E
Provide suggestion	Customer	Customer, Suggestion	Customer request while shopping online	
Send message	Customer	Customer, Message	Customer request while shopping online	
Browse messages	Customer	Customer, Message	Customer request while shopping online	
Request friend linkup	Customer	Customer, FriendLink	Customer request while shopping online	
Reply to linkup request	Customer	Customer, FriendLink	Customer request while shopping online	
Send/receive partner credits	Customer	Customer, CustPartnerCredit, PromoPartner	Customer request while shopping online	
View "mountain bucks"	Customer	Customer, Sale	Customer request while shopping online	
Transfer "mountain bucks"	Customer	Customer, Sale	Customer request while shopping online	

FIGURE 7-19 Deployment diagram for RMO CSMS customer-oriented use cases



Look again at the generic description of three-layer architecture in Figure 7-11 and consider how it might be applied to the use cases in Figure 7-18. Consider that five different actors will interact with the use cases in Groups A–C using different devices and in different operational contexts:

- **Phone sales representative.** Interacts with the customer by phone while using a desktop computer to perform sale and return functions
- **Store sales representative.** Interacts with the customer in person while using a point-of-sale terminal or tablet computer
- **Customer.** Performs functions using a smartphone, tablet, laptop, desktop computer, or other device (e.g., an Internet-capable television)
- **Shipping.** Performs return functions using a tablet or desktop computer
- **Management.** Performs order and shipment functions using a tablet, laptop, or desktop computer

If the domain model is implemented within a single database, the underlying set of use cases can be organized as one data layer interacting with one or more business layers. Use cases could be grouped into business layers in multiple ways. For example, each use case group could be a separate business layer. However, because all of the use cases (Groups A–D) have significant overlaps in users, domain classes, and events, it makes sense to combine them into a single business layer.

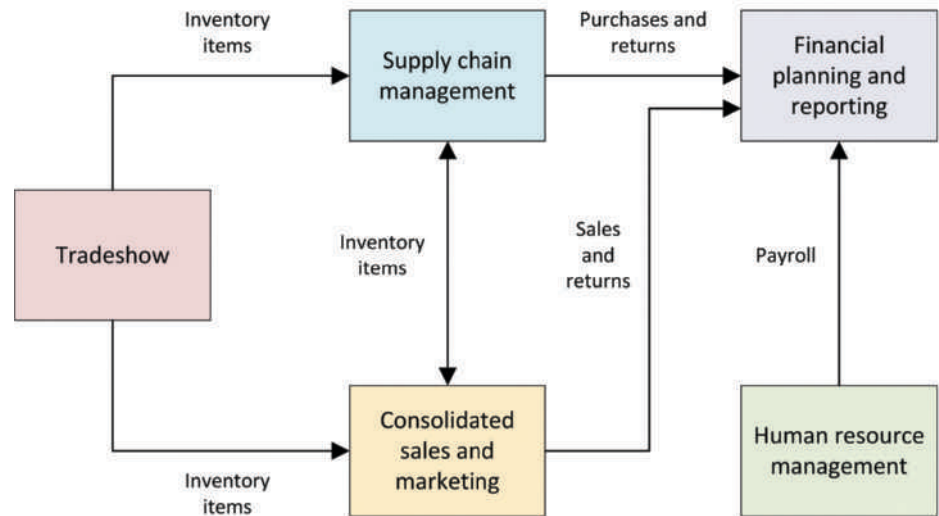
The single business layer interacts with multiple view layers, each optimized for a different combination of user and device. View layers are customized for each user type by omitting connections to irrelevant use cases and by optimizing the user-interface design to workflow and context. For example, the user interface for a phone sales representative would omit Groups B, D, and E use cases and be optimized for rapid data entry using a desktop computer with multiple large displays. In some cases, such as a shipping representative using a tablet or desktop computer, a single view layer can support multiple device types. In other cases, such as customers shopping online, different view layers are required for some devices due to significant differences in device capability or screen size.

Figure 7-19 shows a deployment diagram describing the three-layer architecture for the CSMS use cases in Figure 7-18. Note that the diagram distributes the three layers across two servers—one hosting the data layer and the other combining the business and view layers. Of course, the two servers could be clustered or replicated across multiple locations. The diagram represents one distinct CSMS subsystem in enough detail to enable user-interface, database, and lower-level software design activities to proceed.

■ Application Component Integration

Modern application developers are often faced with the task of integrating legacy systems, purchased application components, third-party SaaS applications, and custom-developed components. Recall from Chapter 2 that RMO's CSMS

FIGURE 7-20 RMO subsystems and data flows



will replace the legacy phone and retail store sales systems, but not the Supply Chain Management (SCM) system. Recall from Chapter 1 that a Tradeshaw System is also under development. Also, like any large company, RMO has other systems, including a SaaS Financial planning & reporting (FPR) system and a purchased Human resource management (HRM) system. **Figure 7-20** summarizes the RMO systems and their interactions.

Transaction information that affects accounting records and financial reports flows into the FPR system from three other systems. If the systems were all internally developed and implemented, the FPR system could directly access the databases of the other systems. But because the FPR is a SaaS application, developers at RMO have no control over its internal implementation. Because it's impractical for a SaaS provider to rewrite their application to access the unique internal systems of every client, most SaaS developers take one or more of three approaches to data inflow from other systems:

- **Batch data export and import.** Data from the source system is extracted and stored in a “neutral” format such as a comma-delimited, Excel, or XML file. The files are stored in an accessible location such as an FTP server and the SaaS system periodically executes a script that copies the file(s) and imports their contents.
- **Data import via Web services.** The SaaS system provides general-purpose Web services for data import. User organizations develop customized data export software to extract data from their systems or databases and pass it to the SaaS system via Web service calls. Data export/import can occur periodically in batches or per update in real time.
- **Direct access to commonly used third-party applications.** When a sufficient number of clients use the same purchased or SaaS software for other purposes, a SaaS provider might develop a specific data export/import utility for that system. For example, many colleges use BlackBoard Learn as a learning management system and Elucian Banner as their student information system. Because each system needs access to the other's data, both vendors have an incentive to develop data export/import software that directly accesses the database of the other system.

RMO currently uses batch data export and import to move financial data from the SCM and HRM systems to the FPR system. RMO-developed scripts export purchase and return transactions from the previous day and store them in an Excel file at midnight. The FPR system is configured to access that file at

2:00 A.M. and add all the transactions to the financial records. The benefits of this approach are low cost and complexity. The drawback is that financial records don't reflect purchasing transactions as they occur. A manager viewing purchasing-related accounts through the FPR system is looking at yesterday's data.

HRM data export to the FPR system is handled similarly, though the export occurs every two weeks to coincide with payroll generation and the file format is XML. The drawbacks and benefits are similar, though the issue of managers accessing "old" data through the FPR system is more significant. Employee time and attendance data is captured daily through the HRM system, but that data isn't shown on RMO financial reports until payroll is generated. Extra accounting steps are thus required at the end of each month to manually adjust accounting records for accrued payroll cost so that financial reports are accurate.

Sharing inventory information among three internally developed systems creates more significant integration challenges. The CSMS will replace the legacy phone and retail store sales systems that generate updates to inventory levels and financial transactions. In theory, the CSMS, SCM, and Tradeshow systems could share the same underlying database, thus eliminating the need to export or import inventory data among them. However, the SCM is a legacy system that RMO would prefer to leave untouched during development of the Tradeshow System and CSMS.

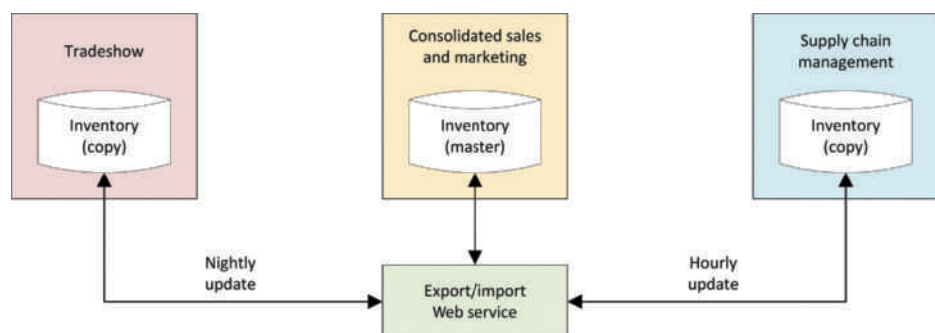
Note the two-way arrow between the SCM system and the CSMS in Figure 7-20. Both systems read and update inventory records and need current and accurate data. So which system "owns" inventory records? The answer is that they both do, and that answer implies that the systems should share real-time access to common data. But implementing that solution would require the new CSMS to use the legacy SCM system's database. Though that solution is technically feasible, it forces new application software to interface with an older DBMS, which may limit some desired capabilities of the CSMS. Also, what about noninventory data used by the CSMS? Will the SCM database be expanded to include that data or will the CSMS access two different databases?

RMO management is reluctant to create too many dependencies between new and old systems. Thus, they've decided to use a separate database to support the CSMS. No changes will be made to the SCM, which will continue to use its current database. Those decisions force a choice concerning inventory data. One of the systems must be declared the system of record for inventory data. A **system of record** maintains a master copy of data that is correct and current. Any other system that maintains a copy of the same data must periodically synchronize it with the system of record.

Figure 7-21 shows the detailed impact of declaring the CSMS to be the system of record for inventory data. CSMS developers will create a Web service that imports and exports information about inventory data for the CSMS database. The Web service format will enable updates over the Internet and provide future extensibility. The Web service will push data updates to the databases for the other two systems and copy any updates from those systems back to the CSMS.

system of record a system or application component that maintains the current and correct master copy of one or more data items

FIGURE 7-21 RMO inventory data synchronization



database. The Tradeshow System accepts inventory level updates nightly and sends back any new inventory item records created by RMO buyers. Updates to/from the CSMS and SCM system are more frequent because current data is more critical to both systems. The SCM system processes product shipments from vendors, and the CSMS needs to know of them quickly so that customers see inventory as it becomes available, and so that shipments for current orders and back orders can be quickly generated. The SCM needs frequent updates of inventory levels due to sales and shipments so that it can generate timely reorders from vendors.

CHAPTER SUMMARY

This chapter described early activities that bridge the gap between systems analysis activities and detailed design activities, such as user-interface design, database design, and internal design of software components. Those bridge activities include describing the system environment and designing its application components. Describing the system's environment requires describing its technical architecture and any interfaces with other systems. Designing the application components requires allocating system functions to components, describing how data is shared among those components, and describing how the application

components are distributed across the technical architecture.

Diagrams developed during these design activities include locations, network, and deployment diagrams. Location diagrams are customized maps that show the physical locations of users and computing devices. Network diagrams show computing devices and the networks that connect them. Network diagrams can be annotated in various ways, including callouts that show embedded software components and protocols. Deployment diagrams show how specific software components are hosted within system software components and computing devices.

KEY TERMS

app	Hypertext Transfer Protocol Secure (HTTPS)	system software
application software	Internet backbone network	three-layer architecture
business logic layer	interoperability	Uniform Resource Locator (URL)
client/server architecture	local area network (LAN)	view layer
data layer	network diagram	virtual private network (VPN)
domain layer	protocol	Web-based application
Extensible Markup Language (XML)	server	Web service
hyperlink	software as a service (SaaS)	World Wide Web (WWW)
Hypertext Markup Language (HTML)	system of record	
Hypertext Transfer Protocol (HTTP)		

REVIEW QUESTIONS

1. What is a server? Do all organizations with information systems own servers? Why or why not?
2. Compare and contrast Internet backbone networks with local area networks.
3. Is the Internet the same thing as the World Wide Web? Why or why not?
4. Describe the parts of a URL.
5. Compare and contrast application and system software. List a few specific examples of each type.
6. Which is more secure, an app or a Web-based application? How or why?

7. List three types of embedded software on a typical smartphone. Of what benefit is embedded software to an application software developer?
8. What is the role of protocols in modern software and systems?
9. List and briefly describe at least three commonly used Web protocols.
10. Briefly define the terms *technology architecture* and *application architecture*. How are they different? How are they interdependent?
11. Describe software as a service. What features distinguish it from application software installed on a personal computing device?
12. How or why are Web services important to designing and constructing modern application software? What decisions does a system designer need to make concerning Web services?
13. In what way(s) is three-layer architecture different from client/server architecture? In what way(s) is it similar?
14. List and briefly describe the function of each layer of three-layer architecture. On what type of computing device is each layer typically deployed?
15. How or why is interoperability such an important consideration in designing and deploying a modern system?
16. List and briefly describe three common types of architectural diagrams developed and used by system designers.
17. Describe the key questions that should be asked and answered when a system designer is describing the environment of a system.
18. How are use cases and related information used by designers when designing application components?
19. What is a system of record? Under what conditions does a designer need to declare one system to be the system of record for a specific type of data? Under what conditions can a system designer avoid doing so?

PROBLEMS AND EXERCISES

1. Investigate the technical and application architecture of a large-scale digital content provider such as iTunes or Google Play services. Update the architectural diagrams in the first half of the chapter to match your chosen provider. Explain any differences from the original diagrams describing the Amazon shopping application.
2. A medium-sized engineering firm has three separate engineering offices. In each office, a local LAN supports all the engineers in that office. Due to the requirement for collaboration among the offices, all the computers should be able to view and update the data from any of the three offices. In other words, the data storage server within each LAN should be accessible to all computers, no matter where they are located. Draw a network diagram that will support this configuration.
3. Consider the differences in physical characteristics among a current smartphone, a current tablet, and a current laptop computer with a 15-inch or larger screen. Also consider the student-facing Web applications you use at your college to access course learning materials, register for classes, and pay college-related bills. For which devices is a unique view layer required? Why? Should any of the applications be constructed as an installable app? Why or why not?
4. Reexamine the decision made in the RMO example at the end of the chapter to make the CSMS the system of record for inventory data. What would be the impact of deciding that the SCM system is the system of record for inventory data? How would Figure 7-21 change?

CASE STUDY

Data Integration at Cooper State University

Cooper State University (CSU) has used a paper-and-pencil survey system to gather student feedback on instructors and courses for over two decades. Paper forms are passed out to students during a class period near the end

of the semester. Class information and survey questions are preprinted on the forms and empty circles are provided for each answer. Students fill in one of the circles for each question with a pencil. Written comments can be provided on the back of the form. The forms are delivered to a processing service in Nebraska, which scans the forms, summarizes the responses, and produces printed reports that

are sent back to CSU along with the survey forms. The reports and survey forms are distributed to department chairs who review them and then distribute them to instructors.

CSU wants to replace the existing system with a modern online survey system. Several vendors market such systems as SaaS Web-based applications. The systems all have similar data integration needs with existing CSU systems, including the following:

- Data about course sections, assigned instructor(s), and registered students must be imported from either CSU's online learning management system (LMS) or its student information system (SIS)—the same system that stores grades and transcripts.
- A survey for a specific student and course section must be accessible as a hyperlink from within the course section's Web page in CSU's online LMS. Instructors may configure the LMS to record whether a student has completed the survey in the grade book and provide points toward the course grade as a reward for completing the survey.
- Access to the survey system should be restricted to CSU instructors, staff, administrators, and students using the same user ID and password they use to access other CSU applications. CSU uses a Central

Authentication Server (CAS) to store usernames and passwords. Other CSU systems interact with a CAS Web service when they need to authenticate a user trying to access the system.

- Students view their final course grades via a Web application that is part of the SIS. CSU wants to delay students' ability to view their final grades by one week unless that student has completed surveys for all courses in which they are registered.

Complete the following tasks:

1. Draw a network diagram that summarizes the systems described above; devices used by administrators, staff, instructors, and students; and the network connections among them. Assume that the current survey system will be replaced by a new Web-based system. Assume further that each existing CSU system is hosted on its own server.
2. Which systems need access to current data regarding course sections, their assigned instructors, and registered students? Select a system to be the system of record for that data and justify your decision. Draw a diagram similar to Figure 7-21 to summarize all required synchronizations.

RUNNING CASE STUDIES

Community Board of Realtors®

The Community Board of Realtors Multiple Listing Service is a small system with limited requirements. In Chapter 3, you identified a complete list of use cases. Using the results from your earlier work, do the following:

1. Discuss the requirements of this system for mobility devices. What use cases would be best

utilized on a mobile device? What use cases would be best with a desktop user interface?

2. Should this application be architected and deployed as a Web application? What would be critical factors to consider if it wanted to deploy this application as a Web application?

The Spring Breaks 'R' Us Travel Service

Complete the following tasks:

- Develop a simple network diagram that includes all four SBRU subsystems (resort relations, student booking, accounting/finance, and social networking) assuming that all server-side hardware and software resides at the SBRU home office.
- List the advantages and disadvantages of hosting all server-side hardware and software at the SBRU home office compared with using a large cloud service provider such as Amazon or Google.

- Answer the following questions:

1. Assuming that external hosting costs are roughly equivalent to the cost of operating, staffing, and maintaining a server room at the home office, do the benefits of external hosting outweigh the costs?
2. Does your answer to question #1 change if external costs are 50 percent greater than operating, staffing, and maintaining a server room at the home office?
3. Does your answer to question #1 or #2 change if SBRU expands its market to include Europe and South America?

On the Spot Courier Services

Refer to earlier chapters for the functional details of this application. Large delivery companies like UPS and FedEx use highly customized portable hardware devices to enable their drivers to track pickups and deliveries and to inform drivers of routing changes and other events to which they must respond. As a smaller company, On the Spot Courier services doesn't have the luxury of developing and deploying a customized hardware device for its drivers.

Sandia Medical Devices

Based on the discussion of hardware, Internet, and software technology trends in this chapter, it should be clear to you that the Real-Time Glucose Monitoring (RTGM) system is an interesting combination of older and newer technology. Except for the interface to software and data on mobile phones, the server-side portions of the system are a relatively traditional business-oriented application that can be implemented by using old-fashioned technology. What makes the RTGM system “new” are its client-side functions, including the automated collection of glucose levels, the regular transmission of that data to servers, the integration of communication between patients and health-care providers, and the integration of those functions within software installed on a portable device that can be carried in a user's pocket.

With that in mind, answer the following questions. You may need to do some additional research to fully address them.

1. Should the client-side software be deployed as a browser-based application or as an installable app? Be sure to consider such issues as client/server communication requirements and frequency, user-interface quality, and portability across devices and operation systems.
2. Which (if any) social networking capabilities might make a useful addition to the RTGM system? Be sure to consider the HIPAA requirements described for this case at the end of Chapter 6.
3. When recorded glucose levels generate high-priority alerts, physicians or other health-care

1. Can it deploy similar functionality by having drivers install an app on their smartphones?
2. What are the pros and cons of this approach as compared with a customized hardware device?
3. Is this a feasible alternative given the wide variety of phones that drivers might own?
4. What embedded software on a smartphone will form the app's technology environment?

providers initiate direct contact with the patient. An ordinary phone call over the cellular phone network is one way to support direct contact. Because any client-side device used with the RTGM system must be fully Internet-capable, an Internet telephony application, such as Skype, is another possible way of supporting synchronous voice or video communication with the patient. Should Skype or a similar Internet telephony application be used with the RTGM system? Why or why not? If such an application is used, should it support video? Why or why not?

4. Data mining is an increasingly important technique for medical research. The ability to scan medical records of large numbers of patients over extended time periods enables researchers to better evaluate the effectiveness of drugs and therapies, more accurately connect disease risk levels to specific patient characteristics, and identify patterns of transmission or occurrence, progression, and treatment response for rare diseases and conditions. What types of medical research might be enabled or better supported by the data collected by the RTGM system? Would your answer change if the database were extended to include additional information that might be gathered from the patient's mobile phone (e.g., location information when each glucose level was captured, size and content of the patient's contact list, call history, and the volume of text messages and Internet browsing activity)?

FURTHER RESOURCES

Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice (3rd ed.)*. Addison-Wesley, 2012.

Russ White and Denise Donohue, *The Art of Network Architecture: Business-Driven Design*. Cisco Press, 2014.

Thomas Erl, Ricardo Puttini, and Zaigham Mahmood, *Cloud Computing: Concepts, Technology & Architecture*. Prentice-Hall, 2013.

CHAPTER EIGHT

CHAPTER OUTLINE

- ▶ Understanding the User Experience and the User Interface
- ▶ Fundamental Principles of User-Interface Design
- ▶ Transitioning from Analysis to User-Interface Design
- ▶ User-Interface Design
- ▶ Designing Reports, Statements, and Turnaround Documents

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- ▶ Explain the concepts of user experience, user interface, and usability
- ▶ Describe the metaphors that can be used to assist in user-interface design
- ▶ Describe important characteristics of human-interface objects that affect usability
- ▶ Discuss the important principles of navigation through a software application
- ▶ Explain how a storyboard can be used to help with user-interface design
- ▶ Describe important guidelines in user-interface design for desktop applications
- ▶ Describe important guidelines in user-interface design for mobile devices
- ▶ Design printed and on-screen reports appropriate to users' needs

OPENING CASE INTERFACE DESIGN AT AVIATION ELECTRONICS

Bob Crain was admiring the user interface for the manufacturing support system that was recently installed at Aviation Electronics (AE). Bob is the plant manager for AE's Midwest manufacturing facility, which produces aviation devices used in commercial aircraft. These aviation devices provide guidance and control functions for flight crews, and they provide the latest safety and security features that pilots need when flying commercial aircraft.

The manufacturing support system is used for all facets of the manufacturing process, including product planning, purchasing, parts inventory, quality control, finished goods inventory, and distribution. Bob was extensively involved in the development of the system for several years, including the initial planning and development. The information systems team that developed the system relied heavily on Bob's expertise. That was the easy part for Bob.

What particularly pleased Bob was the final user interface. He had insisted that the development team consider the entire user experience from the very beginning. He didn't want just another cookie-cutter transaction processing system. He wanted a system that acted as a partner in the manufacturing process—similar to the way that AE's guidance and control system interfaces acted as a pilot's partner.

The first manager assigned to the project placed a low priority on user-interface design. When Bob asked why user-interface design wasn't a key focus of early

iterations, the manager replied, "We'll add the user interface later, after we work out the internal functions." When Bob insisted that the project manager be replaced, the Information Systems Department sent Sara Robinson to lead the project.

Sara had a completely different attitude; she started out by asking about events that affect the manufacturing process and about cases where users needed support from the system. Bob and Sara conducted meetings to involve users in discussions about how they might use the system, even asking users to act out the roles of the user and the system in carrying on interactions.

At other meetings, Sara presented sketches of screens and asked users to draw on them to indicate the information they wanted to see and options they wanted to be able to select. These sessions produced many ideas. For example, it appeared that many users didn't sit at their desks all day; they needed larger and more graphic displays they could see from across the room. Many users needed to refer to several displays, and they needed to be able to read them simultaneously. Users made sketches showing how the manufacturing process actually worked, and the team used these sketches to define much of the interface. Sara and her team kept coming back every month or so with more examples to show, asking for more suggestions.

When the system was finally completed and installed, most users already knew how to use it because they had been so involved in its design.

■ Overview

In Chapter 6, you learned that user-interface design is one of the activities of the fourth core process, design system components. In actuality, designing the user interface has characteristics of both analysis activities and design activities.

As an analysis activity, user-interface design requires very heavy user involvement. You saw a good example of this in the chapter opening case. Screen layout is a discovery activity and produces deep understanding of the user processes and needs just like other analysis activities. During the analysis activities, developers discuss inputs and outputs early and often with system stakeholders to identify users and actors and the information they need to carry out their business processes. In the following sections, you will learn different ways to think about and to model the user interface to help yourself and the user consider various methods of interacting with the new system.

As a design activity, user-interface design utilizes other analysis models as input, such as the system sequence diagram. It also produces models in the form of sample layouts that are used by programmers to implement the final system. Because few systems operate autonomously or in isolation, designing the interfaces (inputs and outputs) between a system and its users is an important systems design task. Poorly designed interfaces with people can result in a system that operates less than optimally or doesn't fulfill its purpose. In this chapter,

you will learn concepts and techniques that will assist you in designing screens and reports that are rich, engaging, and efficient—elements that create a user-friendly and effective interface.

User-interface design is much more complex than it used to be, even just a few years ago. Users are doing much more with multiple computing devices. User-interface design was initially only concerned with the layout of screens and reports. Today, you must consider the entire user experience—what devices are being used and what the users are doing on those devices. Designing the user interface is dependent not only on the type of device, such as desktop, laptop, tablet, or smartphone, but also on the specific types of devices that are being used, such as Apple or Android or Windows devices and operating systems. In this chapter, you will learn general guidelines and principles that can be applied across all user-interface and user-experience design.

■ Understanding the User Experience and the User Interface

The use of computing devices and applications continues to proliferate into all aspects of our lives. Does the use of these applications make us more knowledgeable or more empowered? Or do they make our lives more complex with added frustrations? The answer to these questions depends not only on the functionality provided, but also on how easy it is to understand and use these applications. As a developer, you must have a deep understanding of users, applications, devices, techniques, and principles to build effective user interfaces.

■ Definitions

user experience (UX) all aspects of a person's interaction with a software application, including actions, responses, perceptions, and feelings

user interface (UI) the set of inputs and outputs that the user interacts with to invoke the functions of an application

The **user experience (UX)** is a broad concept that applies to all aspects of a person's interaction with a product or service. When the product is a software application, UX includes actions, responses, perceptions, and feelings that a person has when he uses or anticipates using the software application. It is important for designers to think about the overall user experience as they consider the design of the new system and particularly the user interface.

The **user interface (UI)** is the set of inputs and outputs that directly involve an application user. It is the part of the system that the user sees and interacts with. Because the user interface is the only part of the system that the users see, to them the user interface is the system. UI design varies widely depending on such factors as interface purpose, user characteristics, and characteristics of a specific interface device. For example, although all user interfaces should be designed for maximal ease of use, other considerations, such as operational efficiency, may be important for internal users who can be trained to use a specific interface optimized for a specific hardware device (e.g., a keyboard, a mouse, and a large high-resolution display). In contrast, a quite different user interface might be designed for a customer-accessible system that assumes a cell phone as the input/output device.

Sometimes developers think the user interface can be developed and added to the system near the end of the development process, but as noted in the opening case, the user interface is much more important than that. The user interface is a major element in the total user experience. Both the considerations related to the user experience and the details of the user interface must be integrated into all elements of the system development. Developers must be aware that everything that the end user sees or does while using the system is part of the user interface and affects the user experience. **Figure 8-1** illustrates the complex nature of the user's interaction with the system.

Because from a user perspective, the user interface is the entire system, to the user the programs, scripts, databases, and hardware behind the interface are

FIGURE 8-1 Elements affecting the user experience



user-centered design design techniques that embody the view that the user interface appears to be the entire system

almost irrelevant. In fact, good system design makes all of these elements transparent to the user. Design techniques that embody this point of view are collectively called **user-centered design**, which emphasizes three important principles:

- Focus early and throughout the project on the users and their work.
- Evaluate all designs to ensure usability.
- Use iterative development.

An early focus on users and their work is consistent with the approach to systems analysis in this text. User-oriented analysis and design tasks are performed as early as possible and are often given higher priority than other tasks. For example, such user-oriented analysis tasks as stakeholder identification and interviews occur early in the project. User interfaces are designed in early iterations, and user-related design decisions drive other design decisions and tasks.

usability the degree to which a system is easy to learn and use

The second principle of user-centered design is to evaluate designs to ensure usability. **Usability** refers to how easy a system is to learn and use. Ensuring usability isn't easy; there are many different types of users with different preferences and skills. If the system has a variety of end users, how can the designer be sure that the interface will work well for all of them? For example, if it is too flexible, some end users might feel lost. On the other hand, if the interface is too rigid, some users will be frustrated.

Furthermore, ease of learning and ease of use are often in conflict. For example, menu-based applications with multiple forms, many dialog boxes, and extensive prompts and instructions are easy to learn; indeed, they are self-explanatory. Easy-to-learn interfaces are appropriate for systems that end users use infrequently. But if internal users use the system all day, it is important to make the interface fast and flexible, with shortcuts, hot keys, voice commands, and information-intensive screens. This second interface might be harder to learn, but it will be easier to use after it is learned.

The third principle of user-centered design is iterative development—that is, doing some analysis, then some design, then some implementation, and then repeating the processes. After each iteration, the project team evaluates the work on the system to date. Iterative development keeps the focus on the user by continually returning to the user requirements during each iteration and by evaluating the system after each iteration.

Jakob Nielson, Donald Norman, and Bruce Tognazzini (see Further Resources) are active researchers in **human-computer interaction (HCI)**, a field of study concerned with the efficiency and effectiveness of user interfaces to computer systems, human-oriented input and output technology, and the psychological aspects of user interfaces. HCI is a broad field with many closely related fields of study, including interaction design, efficiency, ergonomics, societal impacts, psychological influences, and so forth.

Although the actual computing devices that people use are physical entities, the items that appear on the screens are virtual entities. They exist only as images and sounds. Because of this, it is helpful for HCI developers to use metaphors based on real-world physical entities and actions to design a user interface with a positive user experience.

human-computer interaction (HCI) a field of study concerned with the efficiency and effectiveness of user interfaces vis-à-vis computer systems, human-oriented input and output technology, and psychological aspects of user interfaces

metaphors analogies between features of the user interface and aspects of physical reality with which users are familiar

direct manipulation metaphor metaphor in which objects on a display are manipulated to look like physical objects (pictures) or graphic symbols that represent them (icons)

desktop metaphor metaphor in which the visual display is organized into distinct regions, with a large empty workspace in the middle and a collection of tool icons around the perimeter

document metaphor metaphor in which data is visually represented as paper pages or forms

■ Metaphors for Human-Computer Interaction

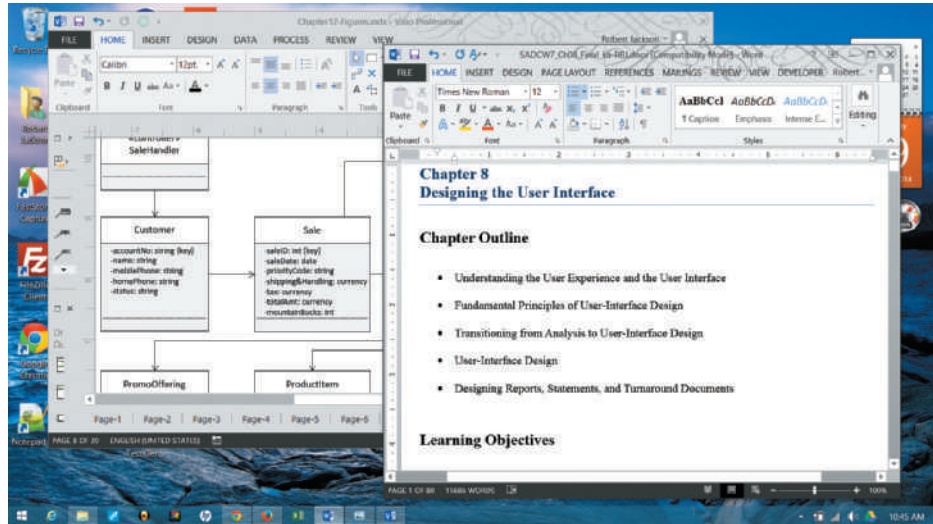
To make computers easier to use and learn, designers of early visually oriented interfaces adopted **metaphors**, which are analogies between features of the user interface and aspects of physical reality that users are familiar with. Metaphors are still widely applied to user-interface design, as described in **Figure 8-2**.

Figure 8-3 is a screen capture of a computer running Windows that illustrates the **direct manipulation**, **desktop**, and **document metaphors**. The entire display is visually similar to the surface of a physical desktop. Icons and pictures for commonly used tools are located on the left and right sides. The icons can be directly manipulated with a mouse or another pointing device. The windows in the center frame are documents that are visually similar to paper pages laid on the surface of a desk.

FIGURE 8-2 Commonly used metaphors for user-interface design

Metaphor	Description	Example
Direct manipulation	Manipulating objects on a display that look like physical objects (pictures) or that represent them (icons)	The user drags a folder icon to an image of a recycle bin or trash can to delete a collection of files.
Desktop	Organizing visual display into distinct regions, with a large empty workspace in the middle and a collection of tool icons around the perimeter	At computer startup, a Windows user sees a desktop, with icons for a clock, calendar, notepad, inbox and sticky notes (the computer interface version of a physical Post-It note).
Document	Visually representing the data in files as paper pages or forms; these pages can be linked together by references (hyperlinks)	The user fills in a form field for a product he or she owns, and the manufacturer's Web site finds and displays the product's manual as an Adobe Acrobat file, which contains a hyperlinked table of contents and embedded links to related documents.
Dialog	The user and computer accomplishing a task by engaging in a conversation or dialog by using text, voice, or tools, such as labeled buttons	The user clicks a button labeled "troubleshoot" because the printer isn't working. The computer prints questions on the display, and the user responds by typing answers or selecting responses from a printed list.

FIGURE 8-3 The direct manipulation, desktop, and document metaphors on a computer display

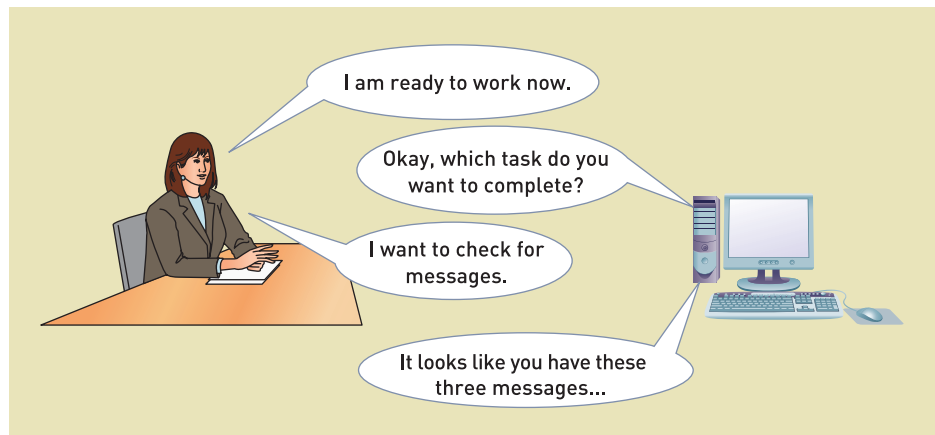


dialogue metaphor metaphor in which user and computer accomplish a task by engaging in a conversation or dialogue via text, voice, or tools, such as labeled buttons

The direct manipulation, desktop, and document metaphors emphasize displayed objects with which the user interacts. The **dialogue metaphor** emphasizes the communication that occurs between a user and a computer, conceptualized as a conversation. In a conversation or dialogue between two people, each person listens and responds to questions and comments from the other person, with the information being exchanged in a sequence. The dialogue metaphor is another way of thinking about human-computer interaction because the computer “listens to” and “responds” to user questions or comments, and the user “listens to” and “responds” to the computer’s questions and comments. **Figure 8-4** illustrates a conceptual dialogue between user and computer.

The dialogue metaphor can be implemented in various ways in user interfaces. A direct approach uses speech generation and recognition over a voice communication channel, as commonly encountered when calling the customer support number of a large company. A computerized voice asks a series of questions, listens for the answer to each question, and responds to the answers. Another implementation of the dialogue metaphor uses questions or instructions displayed by a user through text and responses as well as counterquestions displayed by the computer through text. To minimize the need for user typing, responses to computer questions might be limited to a specific set of possibilities displayed to the user in the form of a list.

FIGURE 8-4 The dialogue metaphor for user-computer interaction



■ Fundamental Principles of User-Interface Design

Many information technology (IT) researchers and practitioners have published articles, books, and Web sites that offer guidance in user-interface design. Many of these guidelines are accepted best practices, having been around for decades. However, the proliferation of new applications, a variety of computing devices, and innovative ways to communicate has required new approaches and principles for user-interface design.

This section provides explanations of some of the most important universal guidelines. It is organized into six major categories: human-interface objects, consistency, discoverability, closure, readability and navigation, and usability and efficiency.

■ Human-Interface Objects

human-interface objects (HIOs) icons and other objects on a screen that can be manipulated by the user and cause some action to occur

Human-interface objects (HIOs) are those objects that appear on a screen that the user can manipulate. Included are such things as documents, buttons, menus, folders, and icons. The following sections discuss several important principles that apply to the design and utility of human-interface objects.

■ Use Human-Interface Objects That Reflect Their Purpose or Behavior (Affordance)

affordance when the appearance of a specific control suggests its function

The use of HIOs is very common in almost all of today's applications. The most common ones we are familiar with are buttons or other small icons. In new versions of the Microsoft Office suite, the ribbons of icons at the top of the page are a good example of useful HIOs whose appearance reflects their function. The technical term for this characteristic is called **affordance**. **Affordance** means that the appearance of a specific control suggests its function—that is, the purpose for which the control is used. For example, a control that looks like a steering wheel suggests that it is used for turning. Affordance can also be achieved by a user-interface control that the user is familiar with in another context. For example, the media player control icons shown in **Figure 8-5** were first widely used on audiotape and videotape players and are still used in such devices as DVD and portable music players. They are widely incorporated into computer interfaces because so many users are familiar with them. Notice in the figure that a **tool tip** is provided. Tool tips are brief instructions that appear when the mouse hovers over a particular tool.

tool tip brief instructions that pop up when the mouse hovers over a control

■ HIOs Should Provide Visual Feedback When Activated

visibility when a control is visible so that users know it is available

feedback some visual or audio response by the system in response to a user action

A second and related characteristic is that HIOs should provide immediate feedback when activated or receiving focus. **Visibility** means that a control is visible so users know it is available; it also means that the control provides immediate feedback to indicate that it is responding. **Feedback** is some visual or audio response by the system in response to some user action. Figure 8-5 is also an example of visibility and feedback. The controls are always visible at the bottom of the window and show immediate feedback on mouseover or mouse click.

FIGURE 8-5 Example of affordance in media player controls

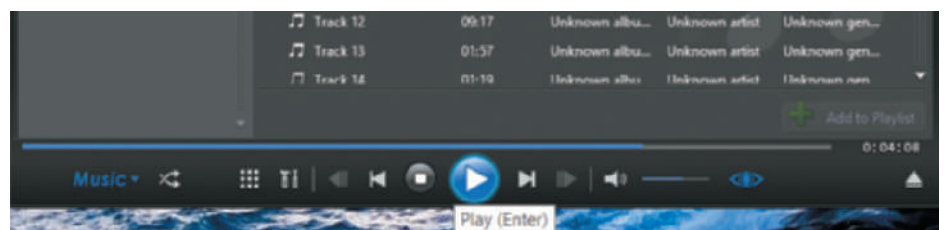
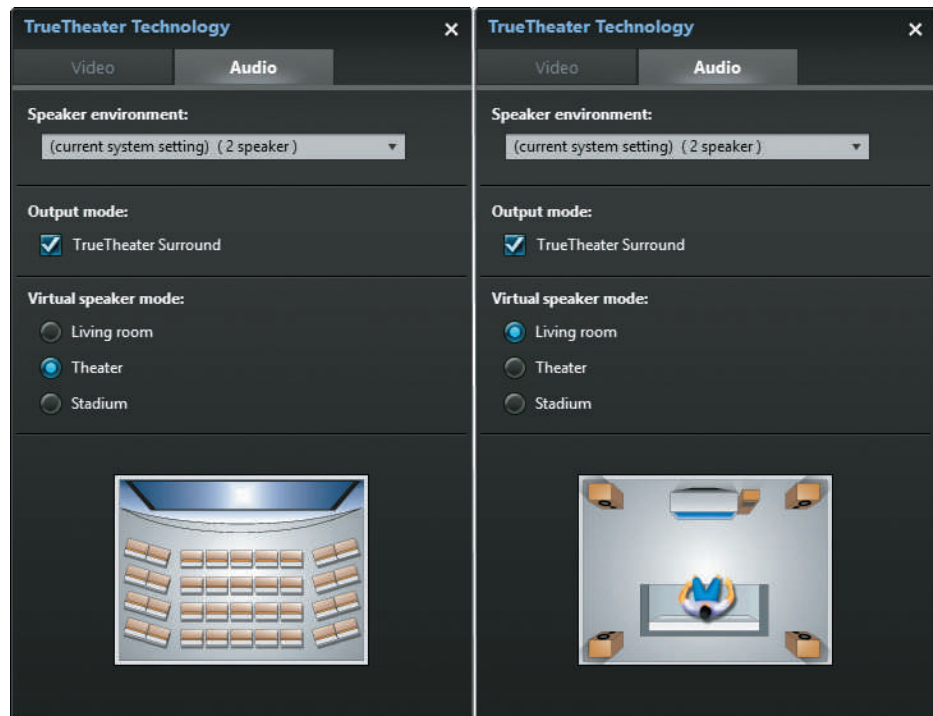


FIGURE 8-6 Example of visibility and feedback



radio buttons a group of selection items that allows only one item to be selected within the group

check boxes a group of selection items that allows either none or many items to be selected within the group

Another example of visibility and feedback is shown in **Figure 8-6**, with a group of **radio buttons** and a **check box**. A group of radio buttons is a control that allows one item to be selected out of the group. A group of check boxes is a control that allows none or many items to be selected. As a radio button is selected, not only does it change its color, but the image also immediately changes. An effective HIO performs some visible action when the mouse hovers over it. It performs some other visible action when it is clicked or double-clicked.

Feedback provides the user with a sense of confirmation and the feeling that a system is responsive and functioning correctly. Lack of feedback leaves the user wondering whether a command or input was recognized or whether the system is malfunctioning.

Visibility and affordance are relatively easy to achieve when the design target is a commonly used platform, such as an iPad, a cell phone running the Android operating system, or a PC running Windows. Such platforms have well-defined user-interface design guidelines and a library of user-interface features and functions that can be reused by application software.

Web user-interface design is less standardized. In some cases, a new icon is used and it is just expected that the user will become familiar with it. Often, these icons are only recognized because of trial and error. Two examples come to mind. The small gear icon has been used to represent settings or options. Another icon, three small bars, is also used. The attempt is to look like a drop-down menu, but it took users a long while to recognize it as a metaphor for settings. Another difficult example is the matrix of small squares to represent applications. This is found on the Google account page. Many users still don't recognize its purpose.

■ Consistency

Consistency can be applied to many different aspects of the user interface, as well as to the application itself. The effectiveness of the user experience is highly dependent on consistency. Users not only expect consistency across the various screens of an application, but now also anticipate consistency across applications. The following sections itemize different levels or areas where consistency should be maintained.

■ Consistency Within and Across Platforms

Consistency within a platform is usually fairly achievable because the tools to build applications and the libraries of existing icons are common and available for all developers. For example, when building Windows applications, it is common to use the standard icons, buttons, check boxes, and other user-interface icons.

However, it is more difficult for developers to achieve consistency across platforms. For desktop computers, there are three common operating environments: Windows, Apple, and UNIX. For smaller mobile devices, there are several different environments, including Android, Windows, Apple, and BlackBerry. Web browsers that execute across multiple platforms help bring consistency to Web-based applications, but they do not solve the problem completely. It is always a challenge for developers who have cross-platform applications to make the user interface as consistent as possible for cross-platform users.

■ Consistency Within a Suite of Applications

Many software organizations have developed and sell a suite of related applications. In many instances, these applications were developed by separate development teams with little communication between teams. The result, of course, is that the user interface for each application has components, and maybe even the entire interface, that are distinct for each application. Organizations that provide a suite should have a coordination group to ensure consistency within the suite.

The Microsoft Office suite is a good example of consistency within the suite. Originally, only the three main components of the suite (Word, Excel, and PowerPoint) had similar user interfaces. However, as Microsoft has acquired other applications and moved them into the suite, it has modified the user interfaces to be consistent. For example, Access, Visio, and Project are not part of the basic suite, but the user interface on these applications now has a consistent look and feel with the ribbon bar across the top and common icons within the ribbon. **Figure 8-7** shows the menu ribbon for three Microsoft applications: Visio, Excel, and Word.

■ Consistency Within an Application

Surprisingly, this concept is probably one of the most violated among developers. Consistency in using the same standard HIOs is quite easily achieved. They are provided in standard libraries. However, when designing such interface items as input forms or data display screens, it is easy to design inconsistent screens. For example, a developer or developers may use different font types, sizes, colors, or bolding for labels and data fields. This may not confuse the user, but it does result in an inconsistent user interface and looks unprofessional.

■ Consistency Versus Continuity

This concept refers to changes occurring in new releases of an application. Almost every application will have upgrade after upgrade with new versions

FIGURE 8-7 Three Microsoft applications showing the menu ribbon

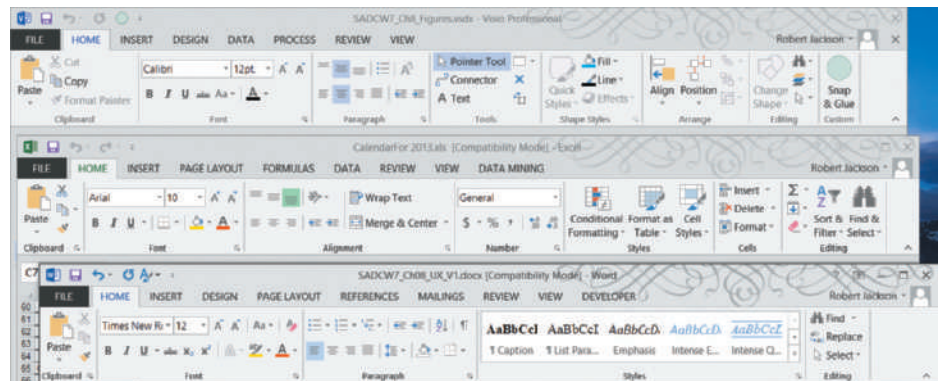


FIGURE 8-8 Home screen for Windows 7 versus Windows 8



continuity maintaining a certain level of consistency over time, across multiple releases

being released periodically. Application **continuity** maintains a certain level of consistency over time across multiple releases. The problem designers have is understanding how to add new functionality while maintaining continuity across new releases. It may not be possible to maintain complete consistency while adding new functions, but continuity should be maintained so that users can easily transition into the new release.

One example that many of us are familiar with was the change from Windows 7 to Windows 8. In this instance, Microsoft wanted to introduce new aspects of Windows to be more compatible with mobile devices. Unfortunately, they ignored continuity and eliminated many aspects that users were familiar with. Microsoft did a quick fix of some of those issued with Windows 8.1, and in Windows 10 brought back even more of the traditional layout, icons, and functions of earlier versions. **Figure 8-8** shows the change from Windows 7 to Windows 8 on the basic home screen.

■ Discoverability

Many applications have more features than can be shown on the initial user interface, especially on smaller devices. Trying to provide an HIO control for every feature would make the user interface too cluttered and complex.

discoverability a feature of the user interface that provides clues to help the users uncover hidden features

One option is to have a limited number of icons, but have all items listed in the menu hierarchy. However, even that approach does not work in all cases. Sometimes it is necessary to have functions or features that exist underneath the initial user interface and must be discovered. **Discoverability** is that feature of the user interface that provides clues to help the users uncover hidden features. It is important that developers consider the discoverability aspect of user-interface design.

One example of features that are not visible or obvious are all the operations that can be performed using the boundary and title bar of a window in Windows 8.1. The only ways to learn about these features is to either watch a tutorial video or to learn by trial and error. For example, moving the window with the title bar to an edge causes it to snap to full size. Another example is closing a metro-style app in Windows 8. To close these full-screen apps, the user must grab the top of the app and pull it to the bottom of the screen. This feature was not obvious, and in fact took many users quite a while to figure it out.

■ Use Active Discovery to Help Find Hidden Features

active discovery a user-interface feature to lead users to discover hidden features

Active discovery is a user-interface feature that leads users into discovering hidden features. There are several methods a designer can use to implement active discovery. One approach is to have a pop-up window appear at application initiation that provides hints and additional features. Another approach is to have a small text box appear as the user hovers over different locations of the screen. This technique is best implemented with a slight delay so that the user is not continually being distracted by the opening and closing of myriad text boxes.

One technique of active discovery used by the Microsoft Office suite is the grouping of similar icons within the ribbon. In the lower-right corner of each group is a small arrow, which indicates that there are more features and functions available. This small clue helps the users find the additional features that are not as visible.

Figure 8-9 returns to the DVD player controls to show these techniques. Notice that some controls are visible, but other controls are hidden. The most important controls are on the main window and are visible and accessible. However, there are icons that open pop-up windows with less-frequently used controls. The tool tips, which appear on mouseover, are also a form of active discovery.

FIGURE 8-9 DVD player with hidden controls and tool tips active discovery

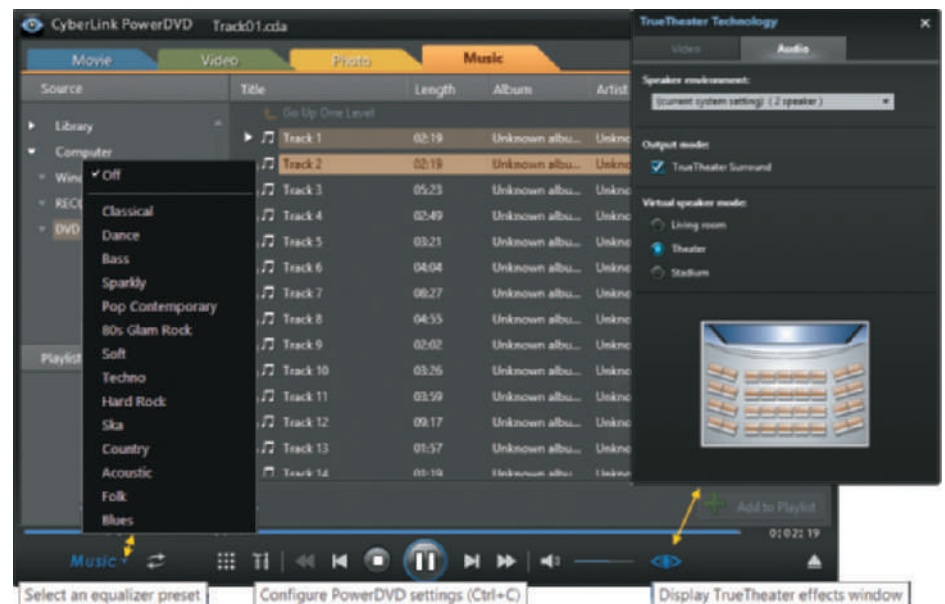
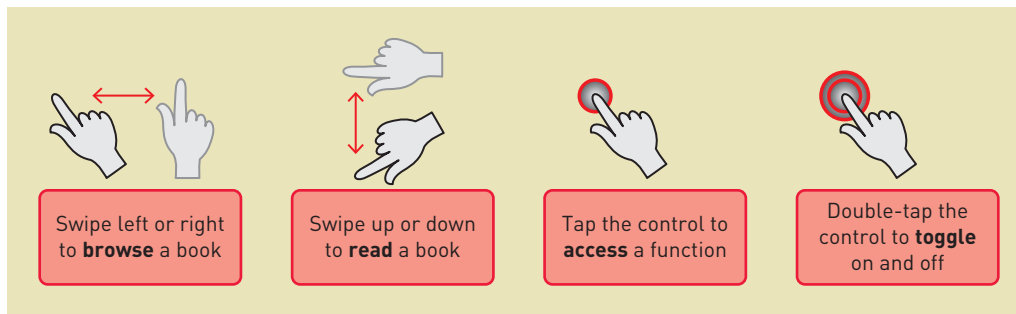


FIGURE 8-10 Help window with diagrammatic explanation



■ Use Visual Diagrams to Help Users Discover Functions or Tools

This third guideline is related to the Help feature of an application. When the user cannot find a desired feature on the user interface, she will sometimes resort to the F1 key or the Help function. Unfortunately, some applications are becoming less user-friendly in this arena. Because most devices are now connected to the Internet, often the F1 key will connect to a server and display the general-purpose Help Web site. In most cases, a user only wants a quick answer to a simple question. A small pop-up window with some instruction or a diagram is still the best solution.

A powerful way to answer a quick query is with a small diagram that displays what action to take. **Figure 8-10** shows a simple diagram that is very expressive of how to perform a desired action.

■ Closure

The idea of closure is based on the dialogue metaphor that was discussed earlier. In most cases, especially for organizational procedures, a use case requires several steps to complete. Sometimes this requires multiple screens; sometimes it can be done with a single screen. In either event, the user interface should have a clearly defined beginning and ending.

■ Provide Closure on Dialogues

Closure is related to the concept of visibility and feedback discussed earlier. Feedback provided on the HIOs is one technique used to provide closure. However, feedback on an individual HIO does not necessarily imply closure. For example, a Save button may change colors and make a clicking sound when it is clicked. However, to ensure closure, it is helpful to have a message pop up saying that the work was saved or the action was completed. The same principle applies to a Cancel button. The Cancel button should provide visual and audio feedback, but refreshing the data fields on a cancel action provides a more declarative closure.

When a use case requires several steps, it is important that the user know that all the steps are completed. One technique to implement this process is to have a Continue button and a Finish button that indicate the end of the process. Sometimes a progress bar is also included and a final message to indicate that the dialogue has successfully finished. Refreshing the final screen to return to a beginning point of the next dialogue also indicates closure.

■ Protect Users' Work

It would appear that the principle to protect a user's work is absolute. However, it applies to a broader set of circumstances than many developers consider. It is common with current applications to always ask the user if she wants to save her work before closing the application. Many applications will also do automatic

saving of a document every few minutes. Another common feature is to inform the user when she tries to save a newer version of a file on top of an older one. These are effective techniques.

However, how many of us have filled out an online form and had to start over again because of either an error or the need to change an input value on the original form? For example, when purchasing concert tickets or airline reservations, if you only want to change a date or time, often the application will require you to complete the entire form for every change. That approach does not conform to the principle of saving the user's work. As much as possible, the user interface should capture and remember the data entered by the user.

■ Make Actions Reversible (Undo)

Users always make mistakes and often need to undo a recent step. Both the application and the user interface must support this reverse function. Users need to feel that they can explore options and take actions that can be canceled or reversed without difficulty. This is one way that users learn about the system—that is, by experimenting. It is also a way to prevent errors; as users recognize they have made a mistake, they cancel the action. In addition, designers should be sure to include Cancel buttons on all dialog boxes and allow users to go back a step at any time. Finally, when the user deletes something substantial—a file, a record, or a transaction—the system should ask the user to confirm the action and, where possible, delay implementing the action.

■ Readability and Navigation

The concept of readability and navigation are particularly important today, yet can be difficult to support. Though large desktop screens usually are quite readable, readability with small mobile devices is often a serious problem. Large desktop screens also have a lot of space for clues about navigating through various screens of an application. The small screens on mobile devices often require that the navigation tools be hidden or partially hidden.

■ Text Should Be Readable (Type, Size, and Color)

There are two primary considerations when designing text to be displayed: the age of your target population and the devices that will be used. Older populations normally require large fonts with clear distinction from the background. Smaller mobile devices limit the amount of text that can reasonably be displayed, so both the type of font and the organization of the text are important. One helpful consideration is that the actual data to be displayed is more important than the labels. Of course, the data must be labeled sufficiently to be understood, but the actual transfer of information occurs with the changing data, not the labels.

The font type is also important. Fancy fonts are frequently hard to discern. Simple sans-serif fonts tend to be better for online reading. **Figure 8-11** illustrates some of the issues with trying to be too fancy with fonts. This Web site advertisement is way too busy. The font colors on the blue background make some parts almost impossible to read. The fancy script lettering is also very difficult to read.

■ Navigation Should Be Clear

Navigation through an application or a Web site depends heavily on the organization of the user interface. The user interface can be organized in a deeply hierarchical vertical fashion or a shallow approach with a flat horizontal span of forms or pages. The vertical approach has fewer menu items at the top level, but requires multiple levels of submenus. The horizontal approach has many menu items at the top level and fewer levels of submenus. The organization of the application or Web site is then reflected in the menu hierarchy provided.

FIGURE 8-11 Sample fonts with readability problems



Either approach works. Each has advantages and disadvantages. Generally, flat designs allow users to see more options quickly. However, deeper menu hierarchies often allow a more organized search approach. In the “depth versus breadth” design, the most important issue is balance. Grouping the various options in logical categories also helps users find their way through the site.

Mobile devices present a challenging platform. Due to the limited screen size, it is often not possible to show all the navigation options. Often the navigation objects are semihidden and must be retrieved to navigate to another page or form.

■ Always Allow a Way Out

This concept applies both to navigation through an application and doing work within a particular page or form. When the user is entering data, he should be able to immediately escape from that form or page without changing the system. Many input forms have a Cancel button to immediately back out of the specific form.

One common technique to allow the user to navigate back out of a location within an application is to display breadcrumbs on the page. **Breadcrumbs (navigation)** is the technique of displaying the sequence of pages traversed to the current page. For example, *RMOHome > Women's clothes > WinterCoats* indicates that the user has traversed three levels down to the winter coats page. Each item in the list is a hot link that will take the user back to that location. This provides both a view of the navigation path and a way to immediately back out and return to any previous level.

■ Usability and Efficiency

One of the primary design considerations in UI design is how to increase user efficiency and effectiveness. In other words, the entire user experience must be considered to make an application effective and usable. The following items are only a few issues that should be addressed.

breadcrumbs (navigation) the technique of displaying the sequence of pages traversed to allow easy backing out to a previous page

■ Provide Shortcut Keys to Support Experienced Users

Software applications normally must be prepared to support users at many different levels of competency and training. Novice users need clear navigation through sequences of steps to complete a task. More advanced users want to complete a task in the fewest steps possible. Hence, user interfaces must be designed to support users across a wide range of capability and expertise.

User interfaces designed primarily for novices are often an annoyance and an impediment to experienced users' productivity. Users who work with an application repeatedly or for long time periods want shortcuts for frequently used functions, which minimize the number of keystrokes, mouse clicks, and menu selections required to complete tasks. Examples include voice commands as well as shortcut keys, such as Windows keyboard sequences Ctrl+C for copy and Ctrl+V for paste. It is always a good idea for application designers to include standard shortcuts when available or build their own for frequently repeated functions.

■ Design Error Messages That Provide Solution Options

A good user-interface design anticipates common errors and helps the user to avoid them or correct them immediately when they occur. One way to do this is to limit available options, presenting the user with only valid options for a specific point in a dialogue. Adequate feedback, as discussed previously, also helps reduce errors.

When the system does find an error, the error message should state specifically what is wrong and explain how to correct it. Consider this error message that occurs after a user has typed in a full screen of information about a new customer:

The customer information entered is not valid. Try again.

This message doesn't explain what is wrong or tell the user what to do next. Furthermore, what if the system clears the data-entry form after this message appears? The user would have to reenter everything previously typed but would still have no idea what was wrong. A better error message would read more like this:

The date of birth entered is not valid. Check to be sure only numeric characters in appropriate ranges are entered in the Date of Birth field.

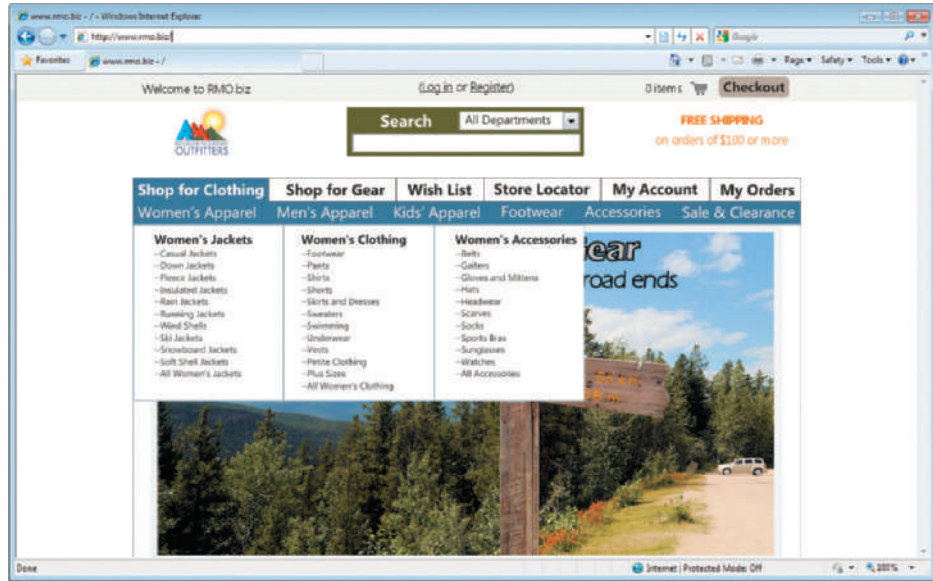
The system also should streamline corrective actions. For example, if the user enters an invalid customer ID, the system should tell the user that this has occurred and then place the insertion point back in the Customer ID text box.

■ Design for Simplicity

The final point is the basic KISS (Keep It Simple Stupid) principle. The simpler the user interface, the better the user experience will be. Fewer mouse clicks is always more productive. Fewer levels to navigate through increase speed of access. Straightforward paths to highly used forms or pages is best. Developers should always be asking themselves whether there is a simpler way to design the navigation and the screens.

Figure 8-12 shows the home page displayed when a customer views the RMO Web site. The form includes two menu bars near the top that group related functions within the same part of the page. If the user points to Shop for Clothing, a submenu is displayed immediately below the menu item and the menu text "Shop for Clothing" changes to white text with a blue background. Labels for the menu items are widely spaced in an easy-to-read font. Except for the logo and picture, the page uses a few complementary colors. The title is positioned near the top of the picture in bold letters. A portion of the title can be seen from behind the drop-down menus.

FIGURE 8-12 RMO home page showing simple design and navigation



■ Transitioning from Analysis to User-Interface Design

The foundation for user-interface design is built from the use case definitions as described in Chapter 3. You learned in earlier chapters how to document the interactions between the user and the system using use case descriptions, activity diagrams, and most importantly system sequence diagrams. Most use cases require direct user interaction and become the starting point for a dialogue or form. Use cases usually require the user to input choices and data into the system (such as when making an online order) or may generate outputs in response to a user request (such as when tracking a shipment).

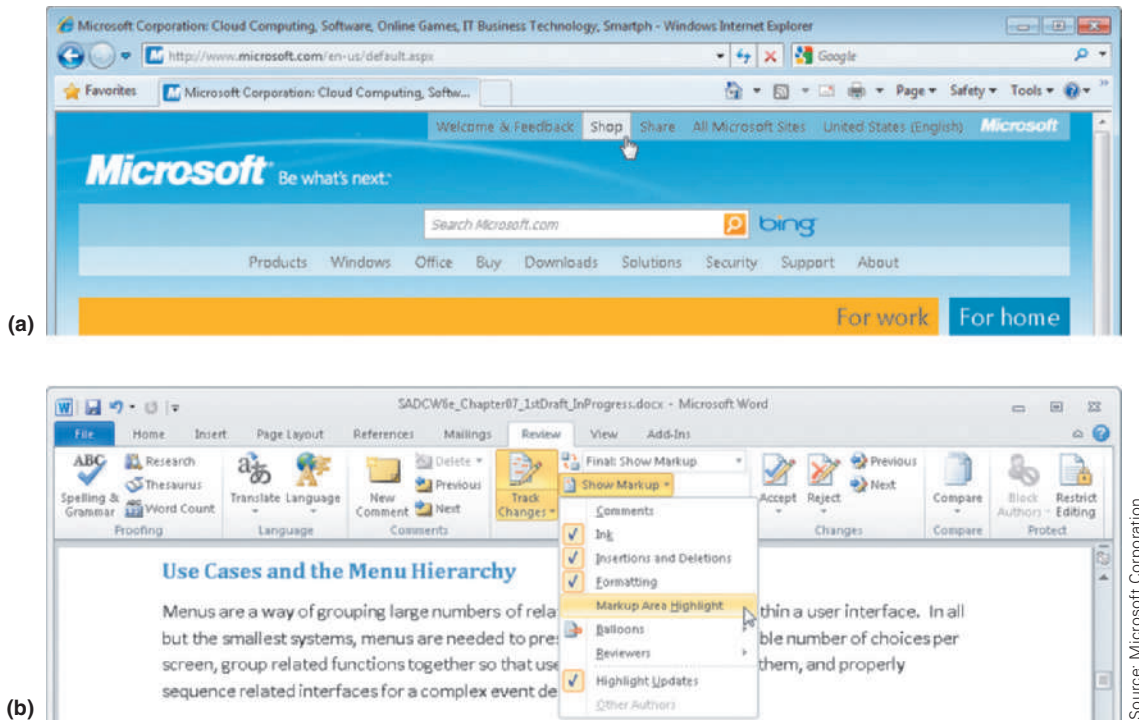
■ Use Cases and the Menu Hierarchy

Menus provide the mechanisms first to organize use cases and second to invoke a particular use case or user dialogue. In all but the smallest systems, menus are needed to present the user with a number of choices per screen, to group related functions together so users can more easily locate them, and to properly sequence related forms to handle complex procedures. **Figure 8-13** shows two different menu styles. In **Figure 8-13(a)**, the mouse pointer is positioned over the Shop item of the upper menu in a Web page. **Figure 8-13(b)** shows a more complex menu design, with three menu levels displayed.

How does a designer decide which use cases and user interfaces to include, which menus are required, and how many menu levels are required? These decisions are driven both by organization of the menus (flat versus deep), by the number of uses cases or menu choices, and by the limits of human cognition. For a typical business system, dividing the total number of interactive use cases by five provides an initial estimate of the number of menus needed that includes all use cases and allows for additional menu items, such as setting options or preferences.

Use cases with common actors and event decomposition or that implement CRUD actions for a specific domain class are good candidates to be grouped into a single menu or related group of menus. For example, consider the RMO CSMS use cases shown in **Figure 8-14**. An initial grouping of these cases by actor and subsystem is a good starting point for menu design.

FIGURE 8-13 Two different menu styles



Source: Microsoft Corporation

FIGURE 8-14 RMO use cases grouped by actor and subsystem

Subsystem	Use case	Users/actors
Sales	Search for item	Customer, customer service representative, store sales representative
Sales	View product comments and ratings	Customer, customer service representative, store sales representative
Sales	View accessory combinations	Customer, customer service representative, store sales representative
Sales	Fill shopping cart	Customer
Sales	Empty shopping cart	Customer
Sales	Check out shopping cart	Customer
Sales	Fill reserve cart	Customer
Sales	Empty reserve cart	Customer
Sales	Convert reserve cart	Customer
Sales	Create phone sale	Customer service representative
Sales	Create store sale	Store sales representative
Order fulfillment	Ship items	Shipping
Order fulfillment	Manage shippers	Shipping
Order fulfillment	Create backorder	Shipping
Order fulfillment	Create item return	Shipping, customer
Order fulfillment	Look up order status	Shipping, customer, management
Order fulfillment	Track shipment	Shipping, customer, marketing
Order fulfillment	Rate and comment on product	Customer
Order fulfillment	Provide suggestion	Customer

© Cengage Learning®

FIGURE 8-15 RMO CSMS use cases grouped into first-cut menus by similar function and user

Menu description	Menu choices (use cases)	Intended user(s)
Shopping cart functions (primary or reserve)	Search for item View product comments and ratings View accessory combinations Switch carts (primary to reserve or vice versa) Fill shopping cart Empty shopping cart Check out shopping cart	Customer
Sale creation	Search for item View product comments and ratings View accessory combinations Create sale	Customer service and store sales representatives
Order shipment	Ship items Manage shippers Create backorder Create item return Look up order status Track shipment	Customer service and store sales representatives
Customer order control	Look up order status Track shipment Create item return Rate and comment on product Provide suggestion	Customer

© Cengage Learning®

Figure 8-15 shows the use cases shown in Figure 8-14 grouped into four menus. Each menu collects uses cases from one subsystem for a customer or internal sales representative. The number of menu choices ranges from four to seven, which won't overload any one menu and may enable multiple menu levels to be displayed at one time. A dialogue design is created for each menu option. You should note that designers often discover missing or incomplete use cases during user-interface design, which results in a brief return to analysis activities to complete the documentation.

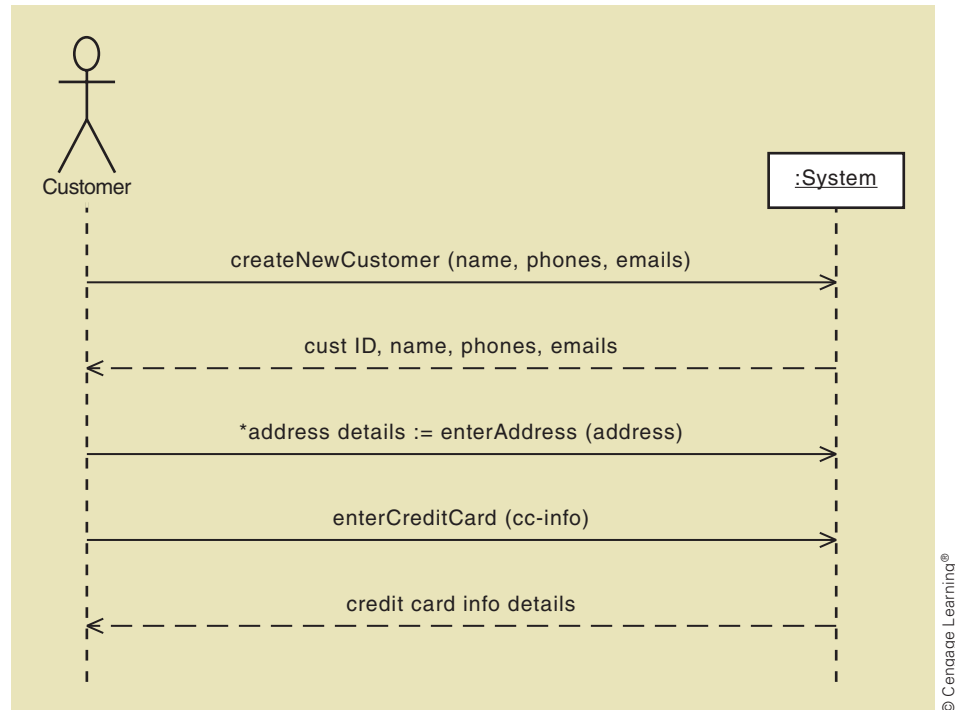
Menus also include options that are *not* activities or use cases from the event list. Many of these options are related to the system controls, such as account maintenance or database backup and recovery, which are discussed later in this chapter. Other added items include help links as well as links to other menus or subsystems.

■ Analysis Models and Input Forms

In Chapter 5, you learned how to document the process flow of a use case using an activity diagram or a system sequence diagram. A system sequence diagram identified individual messages coming from the external actor into the system. The messages included in the sequence diagram represented forms or input screens to pass information into and out of the system. Using a system sequence diagram is a good starting point to identify the various screens and forms that may be needed for the user interface for a particular use case. Figure 8-16 is the system sequence diagram for the *Create customer account* use case, which you first saw in Figure 5-10.

This figure shows six separate data flows across the system boundary. Part of the user-interface design process is to decide how to design the screens for these six data flows. You could design six separate screens; however, notice that there are only three input data flows and three responses reflecting the

FIGURE 8-16 SSD for Create customer account use case



© Cengage Learning®

information that was entered. It might be possible to design the three input screens and use those same screens to display the output of the same fields after the system updates internal files. These kinds of decisions are exactly the ones that need to be made during user-interface design.

Let's develop a first draft of an input screen for `createNewCustomer` (name, phones, emails). The first step is to verify the data that is passed. The place to verify the data is to look at the domain model, to see the attributes of the appropriate classes. Observing the domain model (see Figure 4-25), you can see that there are four fields that require input: name, mobilePhone, homePhone, and emailAddress. Notice that there are two other classes, Address and Account, that will also be used to verify data for the other messages in this use case.

Figure 8-17 illustrates the first draft of the customer screen. It appears that this screen, with some additional messages and buttons, can be used both for inputting data and for displaying the data as it exists within the system.

■ Dialogues and Storyboards

After identifying all required dialogues, the designers must document the dialogues. One approach is to list the key steps the dialogue incorporates, followed by a description of what the user and computer do at each step. The format for writing these steps can mimic the activity diagram described in Chapter 2 or the use case description format in Chapter 4, or it can be more free form.

Designers can also document dialogue designs by writing out how the user and system might interact if they were two people engaged in conversation. Sometimes, the designer can go right from building a dialogue to sketching a series of screens that make the design more understandable. Consider the *Check out shopping cart* use case for the RMO CSMS. A dialogue between the system and user might follow this pattern:

FIGURE 8-17 First-cut input form for createNewCustomer message

- SYSTEM: What would you like to do?
- USER: I'd like to check out.
- SYSTEM: Okay. What is your e-mail address or account number?
- USER: My e-mail address nwells22@gmail.com.
- SYSTEM: Fine. You are Nancy Wells at 1122 Silicon Avenue. Correct?
- USER: Yes.
- SYSTEM: All items in your cart are in stock and can be shipped today. Because your order subtotal is over \$100, you qualify for free UPS ground shipping (3–5 days). Other shipping options include next day (\$35.00), two day (\$20.00), and USPS parcel post (\$11.70, 5–7 days). How would you like your items to be shipped?
- USER: Free UPS ground shipping.
- SYSTEM: We have a shipping address on file for you (1122 Silicon Avenue). Do you want your order shipped to that address?
- USER: No.
- SYSTEM: To what address would you like the order shipped?
- USER: John Wells, 1612 Jefferson Street NE, Albuquerque, NM 87123.
- SYSTEM: Okay, the total charge is \$125.56 (\$117.90 item subtotal plus \$7.66 sales tax). Shall I charge that amount to your credit card on file (a Visa with an account number ending in 0899)?
- USER: Yes.
- SYSTEM: Your payment has been approved, and your order is being prepared for shipment. A confirmation e-mail has been sent to you, and another will be sent with a shipment tracking number when the order is shipped later today. Can I help you with anything else?
- USER: No.

storyboarding sequence of sketches of the display screen during a dialogue

There are many possible implementations of this scenario. Before deciding on a specific implementation, an analyst often uses a technique called **storyboarding**—that is, showing a sequence of sketches of the display screen during a dialogue. The sketches don't have to be very detailed to show the basic design concept. Designers can implement a storyboard with a visual programming tool, such as Visual Basic, or with a presentation program such as PowerPoint.

FIGURE 8-18 Storyboard for the Check out shopping cart *dialogue*

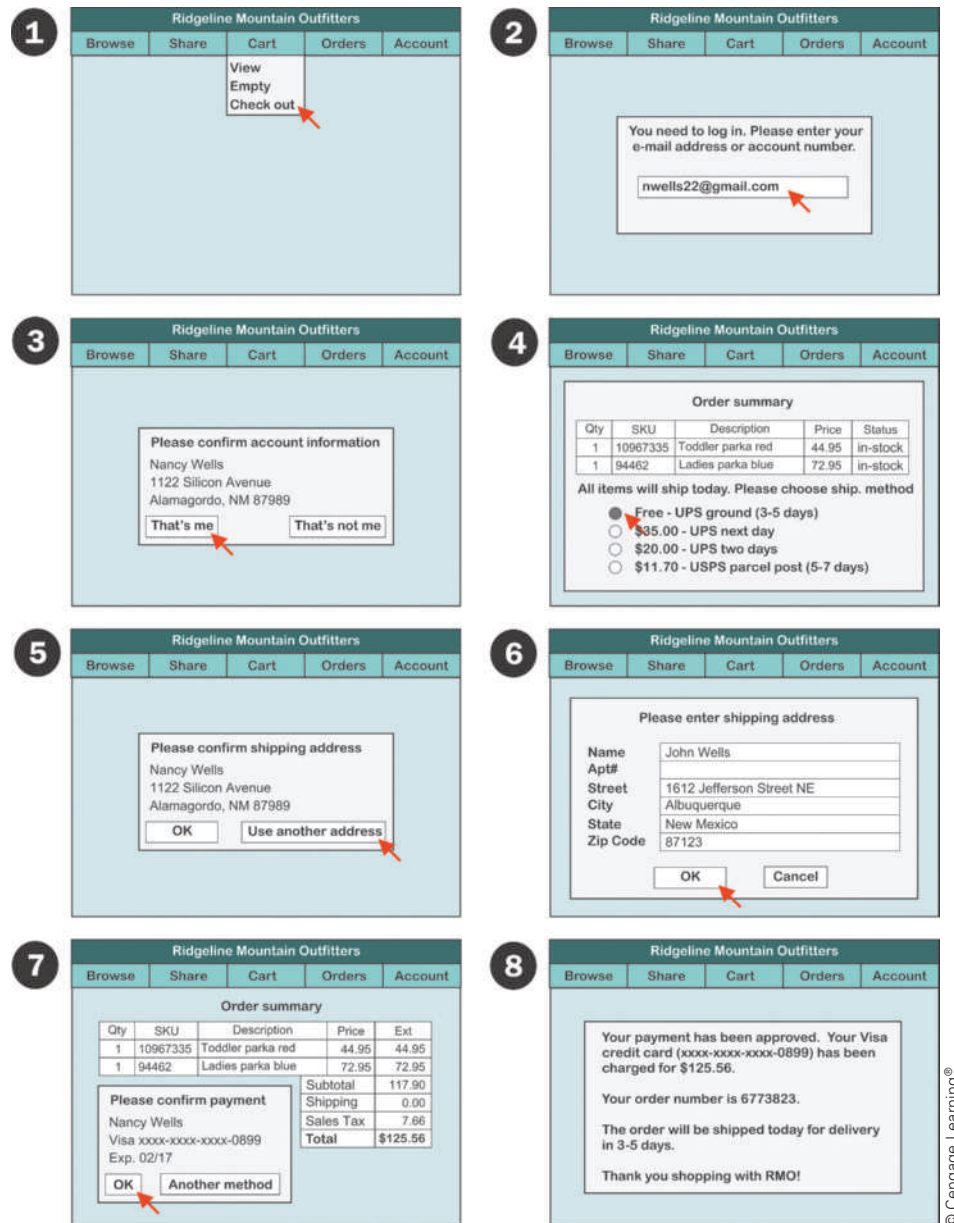


Figure 8-18 shows the storyboard for the dialogue based on the *Check out shopping cart* use case. Though the screen formats are primitive, they are sufficiently detailed to show all the information presented to and entered by the user. The storyboard can be reviewed by users and designers to identify missing or extraneous information and to discuss various options for final implementation, which might be based on a Web page shown on a large display, a traditional Windows or Apple user interface, or the user interface for a mobile device app.

■ User-Interface Design

In the preceding sections, you have learned several basic principles of effective user-interface design. There are many more that could be discussed. The fundamental principles you learned are foundational and apply to all types of user interfaces. In this section, you will learn more of the particulars of designing the user interface for specific devices and configurations.

As you begin the design of the user interface, there are three major considerations that will affect your approach. The first decision, which was probably made at the beginning of the project, is whether the application will be built as a custom, stand-alone software application or whether it will be browser-based. Either approach is viable whether the application is a local, single-computer application or whether it runs over a network or the Internet. If the software is a stand-alone application, then it will utilize the controls and human-interface objects that are available in development libraries for that platform. If it is a browser-based application, then it will need to conform to the controls and configurations provided by the browsers. In some rare instances, the software may need to support both environments.

The second major decision applies to the type of device the application is designed for. Some applications are designed to run only on desktops and laptops. Others execute on tablets, while others are designed exclusively to execute on small devices such as smartphones. In some cases, the software may need to run on multiple devices. In fact, for many types of software, it is desirable that it can be executed on several types of devices. Obviously, this complicates the design of the user interface.

The third major decision applies to the operating system platform. For desktops and laptop computers, the most common operating systems are Windows, Apple Mac OS, and a UNIX derivation such as Ubuntu. For tablets, there are Windows 8 and Windows RT, Apple iOS, Google Android, Amazon Fire, and others such as Ubuntu and BlackBerry. For smartphones, the major operating systems are Google Android, Apple iOS, BlackBerry, Windows Mobile, and others. This decision affects the design because the human-interface objects that are available are different for each platform, especially for stand-alone applications. For Web-based applications, there is a little more commonality, but the browsers are still slightly different for each platform.

As with the fundamental principles, the range of topics for this section is immense. You are encouraged to extend your knowledge of specific platforms through additional books, tutorials, and articles.

■ Desktop and Laptop User Interfaces

There are three areas that will be discussed here with regard to user-interface design: layout and formatting, data entry, and navigation and visibility.

■ Layout and Formatting

Usually, desktops and laptops have large screen displays with adequate space to provide a rich and engaging user interface. Here are several important guidelines:

- **Design screens with purpose.** Each screen should have a primary purpose or use. Do not clutter the screen with unnecessary elements. Allow empty spaces, sometimes called white space, on the screen to keep it simple and easily readable.
- **Consider location and grouping.** Users in the United States read screens top to bottom, left to right. Place important items where they are easily found. Make logical groupings of related items and place them closer together.
- **Ensure professionalism.** Make sure that when people see the screen it looks professional. Align and organize screen objects. Check for poor grammar and spelling errors. Make sure lists have equivalent language constructs (nouns or verbs or adjectives—always the same form).

Figure 8-19 illustrates a sloppy design of an input form with inconsistent labels, inconsistent text box types, poor spacing, misspelled words, and so forth. How would you like to have to fill out data on this form? How would you like to have your name associated with the design of this form?

FIGURE 8-19 *Poorly designed data-entry fields*

The screenshot shows a 'Payment Options' form with the following elements:

- Payment Options:** A group of radio buttons: 'Pay by Credit Card' (selected), 'Pay by Debit Card', 'I want to pay by check', 'Paypal', and 'Send me a bill'.
- CardType:** A dropdown menu currently showing 'Visa'.
- Debit Card:** A dropdown menu currently showing 'Mastercard'.
- Number:** Two text boxes for entering the card number.
- ExpDate:** A text box for the expiration date.
- Paypal email:** A text box for the email address.
- Address:** A text box for the address.
- City:** A text box for the city.
- State:** A text box for the state.
- Zip:** A text box for the zip code.

■ Data Entry

Screens and pages where users enter information require special attention. The overriding objective on data-entry screens is always to minimize errors. The design of the screen can have a big influence on whether the data is error free or error laden. Errors not only reduce productivity, but also can compound errors when the data is used for later processes. There are many ways to make beautiful, effective data input screens. Here are some things to consider as you design the screens:

- **Use controls that minimize keystrokes.** Use controls that do not require keystrokes. List boxes with predefined fields allow quick selection of correct data. Date controls where the user can select a date controls entering correct date and format. Autocorrect fields can also help ensure correct data. Use check boxes and radio boxes when possible. Though these types of controls can reduce productivity, they do minimize errors.
- **Set focus and tabbing.** Set the cursor in the first data-entry box so it is ready to accept data. Make sure the cursor tabs from field to field in a logical fashion.
- **Set font and text box sizes appropriately.** Set font sizes for both labels and entered text so that it is easy for the user to read. Set the size of the data-entry box so that the use of scroll bars is minimized. It is difficult to enter long sentences into a small text box that requires scrolling to view what was entered.
- **Include online edits.** The user interface can be developed to simply accept the entered information, or program code can also be associated with certain fields to perform validations immediately. Some edits are simply logical and can be made locally. Other edits may require validation against a database, and it may not be feasible to perform those edits until the screen is submitted. In any event, it is always best to edit the newly entered data as thoroughly as possible and as soon as feasible.

text box a box that accepts text from a keyboard or speech input

list box a text box with a list of predefined data values

combo box a combination text box and list box that displays a predefined list, but also allows data entry

Figure 8-20 shows a product search screen for RMO. This screen allows the user to search for products using key words and other criteria such as type of products, price ranges, gender, and catalog. In the search group on the lower right, the user can use a **text box** to enter data. There is also a **list box**, which displays a drop-down list of predefined values by clicking on the arrow. A **combo box** is a variation of list box that also allows the user to enter new values if necessary. Contrast this form with Figure 8-19.

■ Navigation and Visibility

When designing for navigation and visibility, the developer must consider not just each screen individually, but the entire application or site. A good site design considers primary as well as secondary paths through the various screens, including both forward and backward movement. Desktop systems should have adequate screen space to allow for clear positioning of menus, hot links, and buttons that are easily located.

FIGURE 8-20 RMO product detail search screen

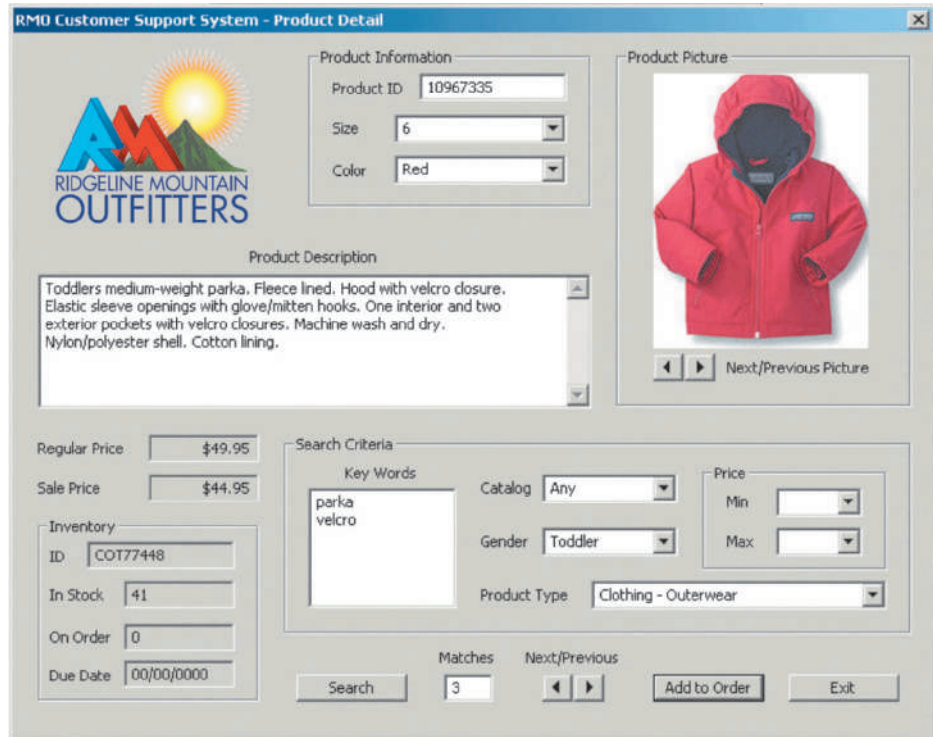
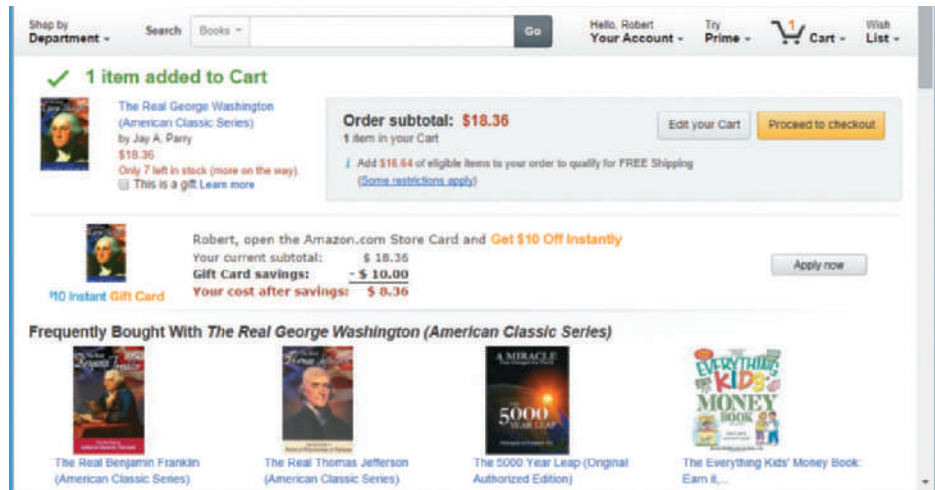


FIGURE 8-21 Online retailer checkout page



Source: Amazon

Figure 8-21 is a sample screen of a Web-based application, where an item was recently added to the shopping cart. Generally, this is a good design, well organized and easily understood. However, it is somewhat limited in its use of navigation buttons. Frequently, a user would like to return to the search page to purchase a related item. Unfortunately, the only way to return to the original list of search results is with the Back button on the browser. A better design would be to include a Continue Shopping link back to the search page.

■ Considerations for Web-Based Applications

When designing the user interface for Web-based applications, all the preceding guidelines still apply. There are also two other important issues to be considered.

The first issue is that every aspect of the user interface must be transmitted over the Internet. With the speeds and bandwidth available today, even large

images and videos are commonly sent to application pages. However, developers still need to be aware of file sizes and page load times. Users may be willing to wait for a video to begin, but they still prefer rapid responses for the initial load. Furthermore, developers should include sufficient error-handling capability to reestablish the dialogue if the connection breaks.

The second major issue is that all the pages of the Web site need to be compatible with all of the potential browsers. The most common browsers today are Internet Explorer, Firefox, Chrome, and Safari. Each of the major browsers implements the primary controls in a somewhat similar fashion. However, each also has unique extensions. Developers must carefully design and thoroughly test Web pages so that they display correctly in each environment. This could require either using controls that are common to all browsers, or testing to see which browser is being used and transmitting the appropriate user-interface code. Here are some additional considerations for design of a Web user interface.

■ Layout and Formatting

As a topic, the layout and formatting of Web pages is extensive. It is impossible to address even a small portion of the issues, guidelines, standards, and conventions that apply to Web user-interface design. Developers who become expert in this arena have a program of continual self-education, reading, and training to design beautiful, engaging, and effective Web pages.

Web applications will be displayed in many different-sized windows. Windows will also frequently be resized on the computer screen. Careful consideration should be given to what happens to the page as it is resized within the window. Poorly designed pages may not resize correctly and may require horizontal scroll bars to view the entire page. Although vertical scrolling is acceptable for most applications, horizontal scrolling is not.

There is one common business practice that user-interface designers continually struggle with: advertising on a Web site. Because the business model for many Web applications includes a revenue string from advertising, designers frequently have to find some way to include sponsor ads. As designers, we will not be able to eliminate all advertising. However, it is a good design practice to place the advertising so that it does not confuse or override the main purpose of the page. This, of course, requires a balance between the revenue stream from the main function of the Web pages and the advertising revenue stream.

In contrast to integrated advertising, **Figure 8-22** is a Web site of only advertisements. It was an interesting marketing idea, but very difficult for any kind of use.

■ Data Entry and User Actions

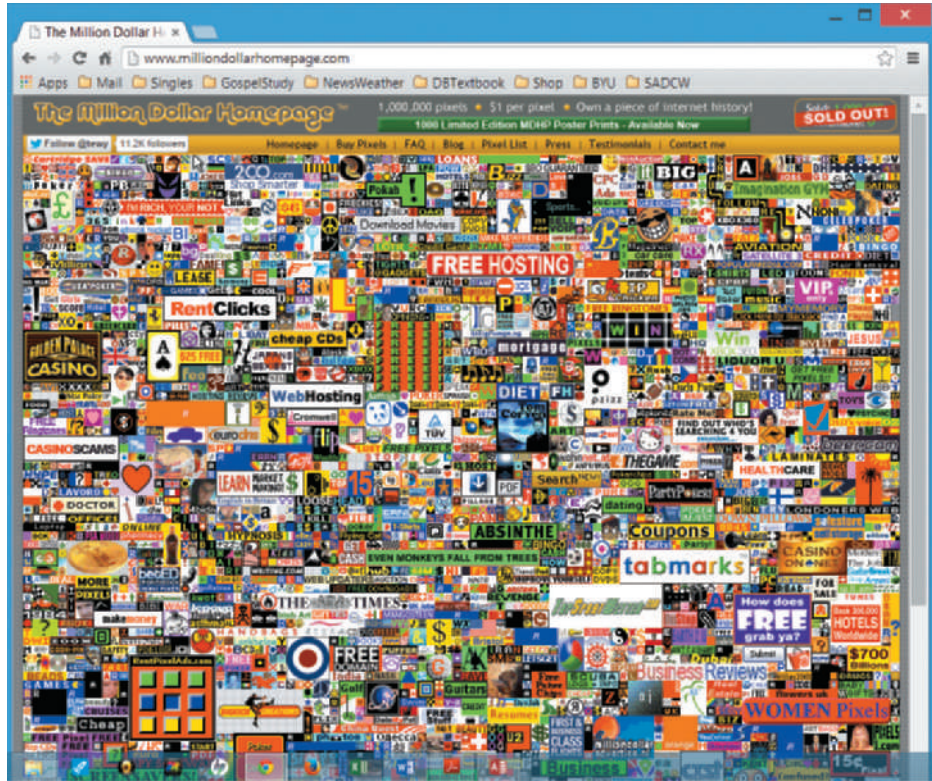
Data entry for Web applications has all the same considerations as for regular desktop apps. With the advent of such capabilities as client-side programming with asynchronous transmission to the server, data-entry validation can occur immediately as it is entered. A primary consideration for data entry is error handling when delays occur over the network.

■ Navigation and Visibility

Standards and conventions for Web applications are in a continual state of flux. Horizontal menu navigation using tabs along the top of the page are currently a popular and effective technique. The guidelines discussed earlier for navigation and visibility also apply to Web applications. It is especially important that navigation links be checked periodically. Web applications frequently have hot links to external sites, which are more susceptible to being broken.

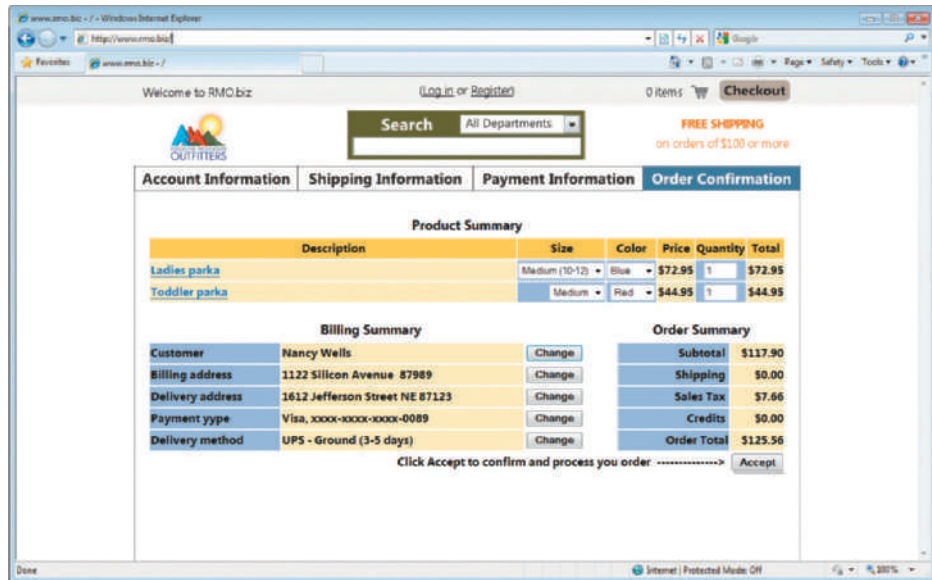
Figure 8-23 is a draft design for the RMO checkout page. Tasks are clear with easily discernable organization and clear instructions. There is plenty of white space to allow for additional instructions or navigation elements if needed.

FIGURE 8-22 Web site of only advertising



Source: The Million Dollar Homepage

FIGURE 8-23 RMO shopping cart checkout page



Source: Microsoft Corporation

Smartphones and Small Mobile Devices

Designing the user interface for smartphones is a very young discipline, and is consequently fluxuating. We will consider some of the issues related to designing the user interface; however, we also recognize that current solutions and approaches will be replaced with better solutions every few months. In addition, because the field is so young, there are few standards or even recognized conventions that are universally applicable. As more experience is gained, you can expect to see guidelines and conventions that are applicable across platforms.

Smartphone apps are usually designed and built for a particular platform. In other words, Android apps must be rewritten to execute on the iPhone. In addition, the user interface is usually tightly connected to the app itself. Some of

mobile responsive designing Web sites so that the pages are responsive to being displayed on small, mobile devices

the earlier principles you learned about multilayer design are not as important for building a smartphone app interface.

The principles you will learn in this section apply to creating a smartphone application. However, all smartphones also have browser programs and thus support Web-based applications. We won't discuss them separately, but the same principles of effective smartphone app user interfaces apply to the design of Web pages viewed on smartphone browsers. Today, all Web-based applications should test the platform that is requesting the page so that it can respond with the appropriate page layout and information. This characteristic is called **mobile responsive design**. Trying to display a full-bodied Web page on the limited real estate available on a smartphone screen almost always ends up completely unusable.

Two primary characteristics of smartphones that influence every aspect of user-interface design are the limited screen size touch screen interaction. Designing for these two characteristics requires a major shift in thinking. Limited screen size dictates that applications and user interfaces must be simple and focused. In addition, almost all navigation and interaction must fit on and be done through the screen. Smartphones have a very limited repertoire of hardware-based buttons and controls, unlike a desktop with a keyboard that has function keys and shortcuts as well as a mouse pointer. However, phone interaction is also changing as voice interaction becomes more common.

■ Layout and Formatting

Given the characteristics of smartphone environment, there are several items affecting layout and formatting that should be noted:

- Each screen should have a single focus or purpose.
- Screens need both portrait- and landscape-view capabilities.
- When appropriate, screens should be allowed to resize without losing screen controls.
- Screen components should be directly touch-manipulated.
- Visible navigation controls should be placed at the bottom of the screen.
- Due to screen swiping, both horizontal and vertical scrolling is allowed.
- Use small pop-up screens when additional information is needed.

FIGURE 8-24 Android smartphone with app



Figure 8-24 illustrates some of the issues related to smartphone apps. There is always an important tradeoff between the work area of the screen, and the navigation elements. In this case, two types of navigation elements are used. The icons will take the user to entirely different pages within the application. The control buttons at the bottom will take the user to the specific functions within the same page. The “Edit” label is a hotlink that will drop down other menu items to allow the user to perform actions on the work area. Thus, even though “Edit” is visible, it contains hidden functions, which requires that the user must pursue active discovery to find all the available actions. It may take the user some time and effort to utilize this smartphone app fully.

■ Data Entry and User Actions

The limited nature of smartphone screens makes data entry and other user actions sometimes difficult to support. Input can occur by typing on a touch keyboard, swiping on the same keyboard, speaking, or other kinds of touching or tapping. Output can be in the form of text, buzzes, beeps, clicks, ring tones, video movies, songs, and vibrations. In other words, smartphone outputs are rich and varied. Smartphone apps should support multiple types of inputs and outputs to take advantage of the capabilities built in to the device. When allowing touch screen actions, developers should plan for the following:

- Be aware of “fat finger” problems where users cannot precisely touch parts of the screen. This means that important control areas should not be too small nor too close together.

- Be aware of accidental touches. Allow a mechanism to reverse without damage. Confirm all actions that delete data.

■ Navigation and Visibility

Navigation and visibility are major issues for smartphone user-interface design. It is generally impossible to keep all navigation controls visible at all times. Developers must pay particular attention to making the application user-friendly in this area of design. Here are a few suggested guidelines:

action bar a common navigation bar with action controls used for smartphones and usually placed at the bottom of the screen

- Design the landing screen (first loading screen) so that it shows clearly how the app is organized and how to get to all the functions. Include a way to return to this screen on all other screens.
- Design the user interface so that there are visual clues for where a task is located in the app.
- Consider using an **action bar** at the bottom (or top) of every screen with common navigation or action controls. If the bar must be hidden, make it easy to find and show.
- Remember that there is no mouseover capability, so such techniques as tool tips are not available.
- Ensure a Back button capability exists, either on the device or as part of the application.

Figure 8-25 is the landing screen for the RMO smartphone app. This screen is almost completely made up of navigation elements to send the user to the appropriate screens. The only image is the small logo at the top. Special attention was given to contrast and layout to ensure readability.

■ Tablets

User-interface design for tablets requires many of the same techniques that are used for smartphone design. Tablets are mobile devices with similar device constraints as smartphones. The primary difference is that there is more screen real estate, so the screen constraints are not as tight.

When considering layout and formatting for tablets, applications must support both portrait and landscape display. Where a smartphone screen can only support a single function or activity, it may be possible to have multiple action areas on a tablet. The small screen size and the lack of physical keyboards and mouse interaction do require that developers still design screen layout carefully.

FIGURE 8-25 RMO home screen on a smartphone

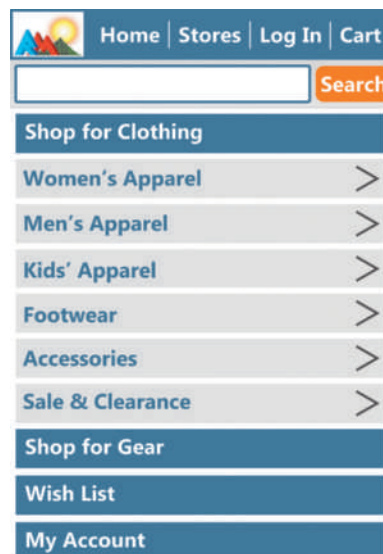


FIGURE 8-26 Application on an iPad tablet



Data entry and user actions are very similar for tablets and smartphones. The lack of a keyboard makes typing somewhat cumbersome, so any technique that reduces actual data entry is helpful. The larger touch screen, however, does permit the use of larger icons and touch areas to alleviate the fat finger problem.

Probably the biggest benefit of the somewhat larger screen is that more navigation controls and menu items can be made visible and remain on the screen. More items can be added to the action bar, and having navigation controls other than the single action bar is possible. **Figure 8-26** is a picture of an iPad with an online store open. The navigation is clear, with a menu on the top and icons for the available categories. Note, however, that the site limits this page to eight categories.

■ Designing Reports, Statements, and Turnaround Documents

Due to the highly interactive nature of today's software applications, there is less need for printing formal reports. In many cases, the screen views that the user sees can either be sent directly to the printer, or a special print view is provided. A print view is usually a simple reformatting of the screen view with the controls, advertising, and extraneous graphics removed. As you work with the users to develop an application, you should continually be asking if a print view is required for each screen that displays information or whether a printed report is the best form for presenting the information.

Modern information systems have made information much more widely available, with a proliferation of all types of reports—paper and electronic. One of the major challenges organizations face today is to compile the overwhelming amount of information available to support managerial decision making. One of the most difficult aspects of output design is to decide what information to provide and how to present it effectively.

■ Report Types

There are four types of output reports commonly provided by an information system:

detailed reports reports that contain specific information on business transactions

summary reports reports that summarize detail or recap periodic activity

exception reports reports that provide details or summary information about transactions or operating results that fall outside a predefined normal range of values

executive reports reports used by high-level managers to assess overall organizational health and performance

■ **Detailed reports.** These contain specific information on business transactions—for example, a list of all overdue accounts, with each line of the report presenting information about a particular account.

■ **Summary reports.** These are often used to recap periodic activity. An example of this is a daily or weekly summary of all sales transactions, with a total dollar amount of sales. Managers often use this type of report to track departmental or division performance.

■ **Exception reports.** These provide details or summary information about transactions or operating results that fall outside of a predefined range of values. When business is progressing normally, no report is needed. For example, a manufacturing organization might produce a report that lists parts that fail quality control tests more than 0.2 percent of the time.

■ **Executive reports.** These are used by high-level managers to assess overall organizational health and performance. They contain summary information from activities within the company. They might also show comparative performance with industry-wide averages.

An example of a detailed report is shown in **Figure 8-27**. When a customer places an order on the Web, the system prints the order information as a confirmation. Of course, a user can always print the Web screen display by using

FIGURE 8-27 RMO shopping cart order report



Ridgeline Mountain Outfitters—Shopping Cart Order

Customer Name: Fred Westing
Customer Number: 6747222

Order Number: 4673064
Today's Date: May 18, 2015

Shipping Address:

936 N Swivel Street
Hillville, Ohio 59222

Billing Address:

936 N Swivel Street
Hillville, Ohio 59222

Qty	Product ID	Description	Size	Color	Price	Extended Price	
1	458238WL	Jordan Men's Jumpman Team J	12	White/ Light Blue	\$119.99	\$119.99	
1	347827OP	Woolrich Men's Backpacker Shirt	XL	Oatmeal Plaid	\$41.99	\$41.99	
2	8759425SH	Nike D.R.I. – Fit Shirt	M	Black	\$30.00	\$60.00	
1	5858642OR	Puma Hiking Shorts	L	Tan	\$15.00	\$15.00	
						Subtotal	\$236.98
						Shipping	\$8.50
						Tax	\$11.25
						Total	\$256.73

Shipping Information:

Shipping Method: Normal 7–10 day

Shipping Company: UPS

Tracking Number: To be sent via email

Email Address: FredW253@aol.com

Payment Information:

American Express MasterCard VISA Discover

Account Number


x
x
x
x
-
x
x
x
x
-
x
x
x
x
-
5
7
8
4

MO YR

Expiration Date 05 / 17

Thank you for your order. It is a pleasure to serve you.
Check back next week for new weekly specials!!

FIGURE 8-28 RMO inventory report



Ridgeline Mountain Outfitters – Products and Items

ID	Season	Category	Supplier	Unit Price	Special Price	Discontinued
RMO12587	Spr/Fall	Mens C	8201	\$39.00	\$34.95	No
Description Outdoor Nylon Jacket with Lining						
Size	Color	Style	Units in Stock	Reorder Level	Units on Order	
Small	Blue		691	150		
	Green		723	150		
	Red		569	150		
	Yellow		827	150		
Medium	Blue		722	150		
	Green		756	150		
	Red		698	150		
	Yellow		590	150		
Large	Blue		1289	150		
	Green		1455	150		
	Red		1329	150		
	Yellow		1370	150		
Xlarge	Blue		1498	150		
	Green		1248	150		
	Red		1266	150		
	Yellow		1322	150		

ID	Season	Category	Supplier	Unit Price	Special Price	Discontinued
RMO28497	All	Footwear	7993	\$49.95	\$44.89	No
Description Hiking Walkers with Patterned Tread Durable Uppers						
Size	Color	Style	Units in Stock	Reorder Level	Units on Order	
7	Brown		389	100		
	Tan		422	100		
8	Brown		597	100		
	Tan		521	100		
9	Brown		633	100		
	Tan		654	100		
10	Brown		836	100		
	Tan		954	100		
11	Brown		862	100		
	Tan		792	100		
12	Brown		754	100		
	Tan		788	100		
13	Brown		830	100		
	Tan		921	100		

the browser's print capability, but that takes more time and report format isn't guaranteed. It is much more user-friendly to provide shoppers with a "printer friendly" order confirmation in addition to a Web-based display.

Figure 8-28 is an example of an internal report based on inventory records. This report includes detailed information; summary information is available on a different report. A control break is used to divide the detailed section into groups. In this example, the control break is on the product item number—called ID on the report. Whenever a new value of the ID is encountered on the input records, the report begins a new control break section. The detailed section lists the transactions of records from the database, and the summary section provides totals and recaps of the information. The report is sorted and presented

FIGURE 8-29 A sample employee benefit report

Survivor Protection

In the event of your death while working for a participating employer, your designated beneficiaries could receive:

Lump Sum Benefits

\$50,000	Basic Life Insurance
\$230,000	Supplemental Life Insurance
\$148,677	Thrift Plan
\$31,686	Tax Sheltered Annuity (TSA) Plan
\$255	Social Security for your eligible dependents
\$460,618	Total*

You have not elected Universal Life Insurance. If you would like more information on this plan, please call 1-800-555-7772.

*Refer to page 7 for additional information on the amount of coverage needed to provide ongoing replacement income.

Accidental Death Benefits
If your death is due to an accident, your designated beneficiaries will receive the above benefits plus:

\$100,000	24-Hour Accidental Death and Dismemberment Insurance
\$100,000	Occupational Accidental Death and Dismemberment Insurance, if the accident is work related

Monthly Death Benefits
If you die before receiving the Master Retirement Plan benefits and you are vested and have a surviving spouse, your spouse may be eligible for a Qualified Pre-Retirement Survivor Annuity.

In addition, your family may be eligible for the following estimated monthly benefits from Social Security, not to exceed a maximum of \$2,591 based on:

\$1,110	for each child under age 18
\$1,110	for a spouse with children under age 16; or
\$1,058	for a spouse age 60 or older

by product. Within each product is a list of each inventory item showing the quantity currently on hand.

Reports that are produced for external stakeholders can consist of complex, multiple-page documents. A well-known example is the set of reports and statements that you receive with your car insurance statement. This statement is usually a multipage document consisting of detailed automobile insurance information and rates, summary pages, turnaround premium payment cards, and insurance cards for each automobile. Another example is a report of employment benefits, with multiple pages of information customized to the individual employee. Sometimes, the documents are printed in color, with special highlighting or logos. **Figure 8-29** is one page of a sample report for survivor protection from an employee benefit booklet. The text is standard wording, but the numbers are customized for the individual employee.

■ Electronic Reports

Organizations use various types of electronic reports, each serving a different purpose and each with its respective strengths and weaknesses. Electronic reports allow flexibility in the organization and presentation of information. Some have detailed and summary sections, some show data and graphics

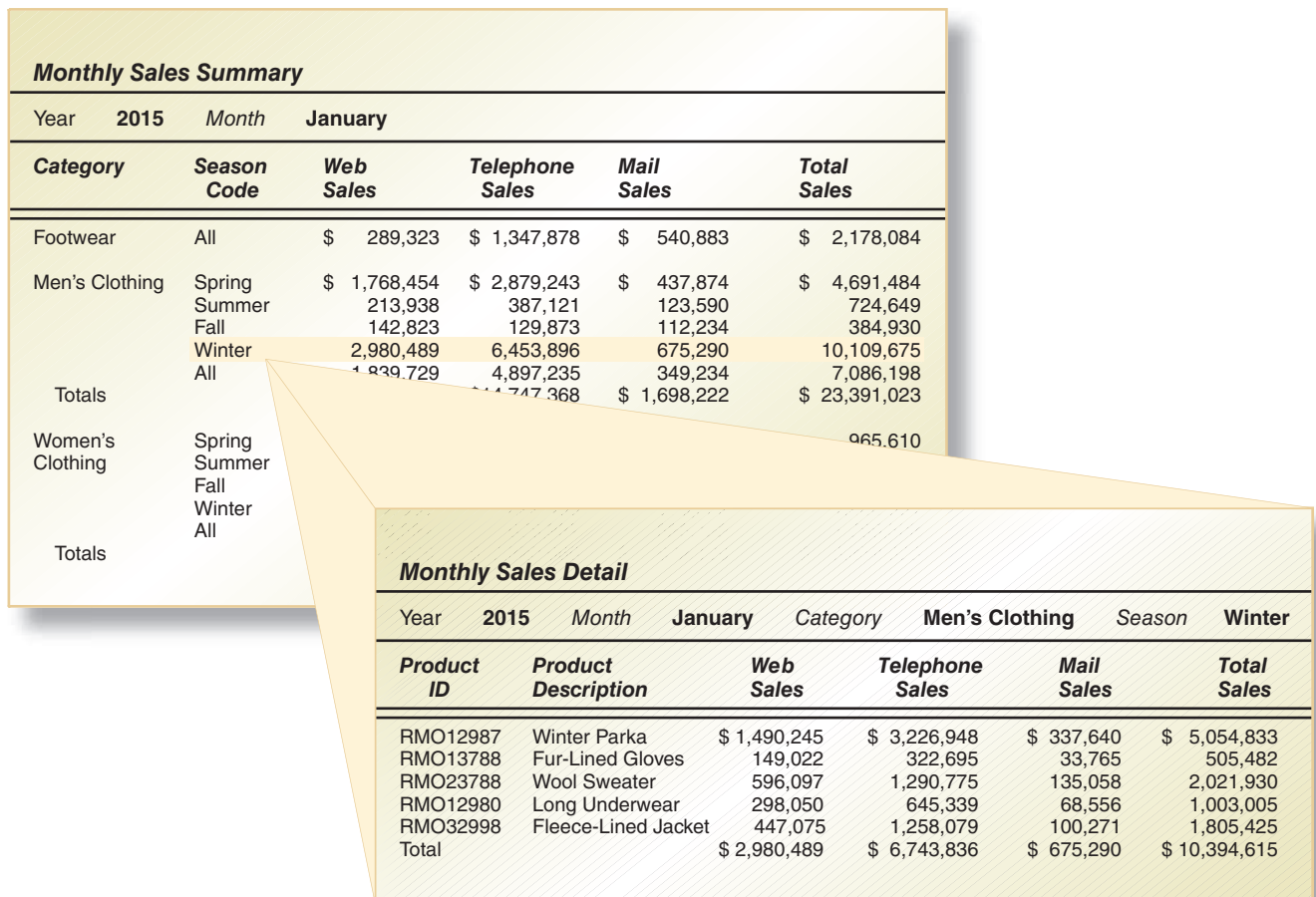
drill down user-interface design technique that enables a user to select summary information and view supporting detail

together, others contain boldface type and highlighting, others can dynamically change their organization and summaries, and still others contain hot links to related information. A primary characteristic of electronic reporting is that it must be dynamic; users should be able to control input to meet the specific needs of a particular situation. For example, an electronic report can provide links to further information. One technique, called **drill down**, allows the user to activate a “hot spot hyperlink” on the report, which tells the system to display a lower-level report that provides more detailed information. For example, **Figure 8-30** contains a monthly sales summary. The report provides sales totals grouped by product category and season. If the user clicks the hot link for any season, a detailed report pops up with more sales data.

Another variation of this hot link capability lets the user correlate information from one report to related information in another report. Most people are familiar with hot links from using their Internet browsers. In an electronic report, hot links can open other information that correlates or extends the primary information. This capability can be very useful in a business report that, for example, links the annual statements of key companies in a certain industry.

Another aspect of electronic reports is the capability to view the data from different perspectives. For example, it might be beneficial to view sales commission data by region, by sales manager, by product line, or by time period

FIGURE 8-30 RMO summary report with drill down to the detailed report



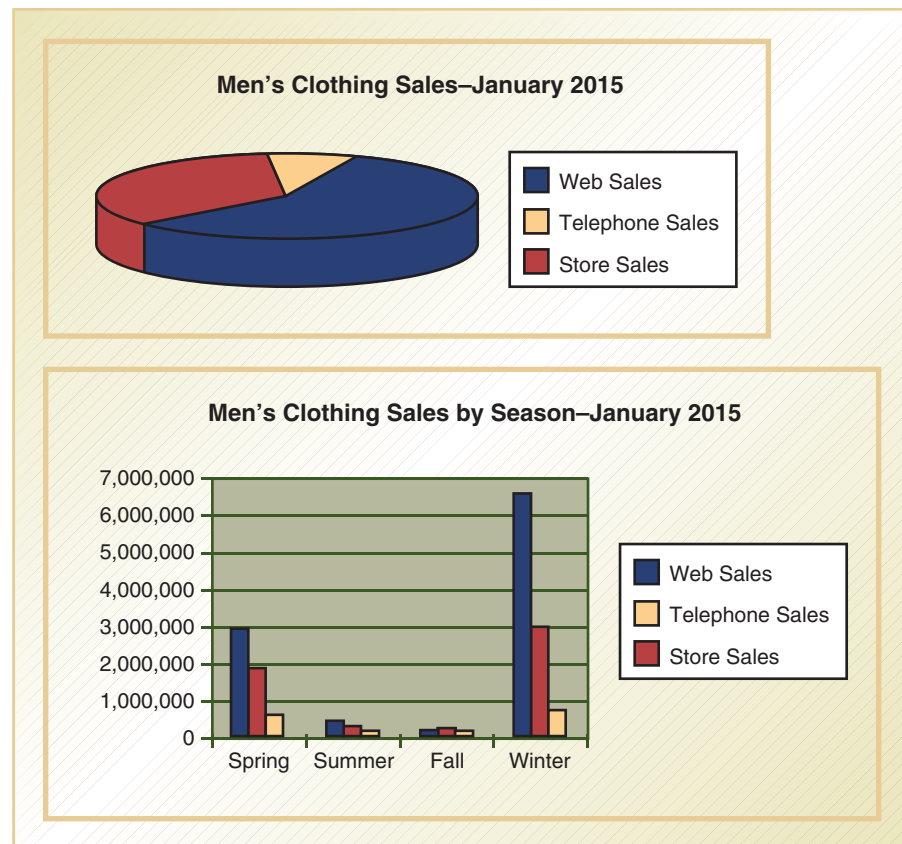
or to compare the current data with last season's data. Instead of printing all these reports separately, you can use an electronic format to generate the different views as needed. Sometimes, long or complex reports include a table of contents, with hot links to various sections of the report. Some report-generating programs provide an electronic reporting capability that includes all the functionality found on Web pages, including frames, hot links, graphics, and even animation.

■ Graphical and Multimedia Presentation

The graphical presentation of data is one of the greatest benefits of the information age. Tools that permit data to be easily depicted in charts and graphs have simplified information reporting for printed and electronic formats. Information is being used more for strategic decision making as businesspeople examine their data for trends and changes. In addition, today's systems frequently maintain massive amounts of data—much more than people can review. The only effective way to use much of this data is by summarizing and presenting it in graphical form. **Figure 8-31** presents a pie chart and a bar graph—two common ways to present summary data.

Multimedia outputs have become available recently as multimedia capabilities have increased in applications. Today, it is possible to see a graphical (possibly animated) presentation of the information on a screen and have an audio description of the salient points. Combining visual and audio output is a powerful way to present information.

FIGURE 8-31 Sample pie chart and bar graph reports



CHAPTER SUMMARY

The primary focus of user-interface design is on the user's needs, requiring major involvement by the user. As such, user-interface design has characteristics of both analysis and design activities. Today's UI design efforts focus on more than building the screens and reports, but also considers the entire user experience. The use of various metaphors, such as direct manipulation, desktop, or dialogue metaphors, helps developers better understand and think about how the users interact with the system.

User-interface design is a complex discipline and usually requires substantial knowledge and experience to understand important influencing factors. Six key factors to consider are human-interface objects, consistency, discoverability, closure, readability and navigation, and usability and efficiency. Each of these categories has several key issues that need to be addressed during design activities.

Input for the user interface comes from the diagrams and models that were developed during the analysis activities. The list of use cases is often helpful to define menus and navigation components. Activity diagrams and system sequence diagrams help identify particular screens and reports that are needed.

A major difficulty of UI design is the many different devices and screen types that must be supported, from large screen desktop systems to miniature smartphone systems. In addition, many different platforms, such as Apple versus Android devices, require multiple interfaces to display correctly on each device.

Although output reports are not used as widely as they once were, there is still the need to produce printed output of some information. Furthermore, today's printed reports need to be more sophisticated by integrating charts and graphs. Users expect sophisticated electronic reports with hot links, which allow drill down to display more detail.

KEY TERMS

action bar	discoverability	radio button
active discovery	document metaphor	storyboarding
affordance	drill down	summary reports
breadcrumbs (navigation)	exception reports	text box
check box	executive reports	tool tip
combo box	feedback	usability
continuity	human-computer interaction (HCI)	user-centered design
desktop metaphor	human-interface objects (HIOs)	user experience (UX)
detailed reports	list box	user interface (UI)
dialogue metaphor	metaphor	visibility
direct manipulation metaphor	mobile responsive	

REVIEW QUESTIONS

1. Why is user-interface design often referred to as dialogue design?
2. What are some examples of the physical, perceptual, and conceptual aspects of the user interface?
3. What are the three metaphors used to describe human-computer interaction?
4. A desktop on the screen is an example of which of the three metaphors used to describe human-computer interaction?
5. What is the technique that shows a sequence of sketches of the display screen during a dialogue?
6. What are some of the input controls that can be used to select an item from a list?
7. What two types of input controls are included in groups?
8. What are the different considerations for output screen design and output report design?
9. What is meant by the user experience? How does the user interface relate to it?
10. What is meant by usability and why is it important for user-interface design?

11. Describe each of the four metaphors for user-interface design.
12. What are human-interface objects? Give some examples.
13. Describe affordance and provide an example from your experience.
14. What is meant by a tool tip? How does it show up?
15. Give several examples of feedback. Why is it important?
16. What is the difference between visibility and affordance?
17. Identify and describe three levels of consistency.
18. What is the difference between consistency and continuity?
19. What is active discovery? Give two examples of how to implement it.
20. What is closure? Why is it important?
21. How do breadcrumbs work? Show an example from one of your activities, either in an app or a Web site.
22. What is the difference between a flat menu design and a deep menu design?
23. How can use case diagrams help in the design of the user interface?
24. How can system sequence diagrams help in the design of the user interface?
25. What is storyboarding?
26. Describe three major decisions that must be made early in the project that directly affect user-interface design.
27. What is the difference between a combo box and a list box?
28. What are two major considerations when designing the user interface for browser-based systems?
29. What does it mean for an application to be mobile responsive?
30. What is meant by the “fat finger” problem? How does it affect user-interface design?
31. What is an action bar? Have you noticed one on an app? Describe it.
32. What is the difference between a summary report and an exception report?
33. What does drill down mean? How is it displayed?

PROBLEMS AND EXERCISES

1. Think of all the software you have used. What are some examples of ease of learning conflicting with ease of use?
2. Visit some Web sites and then identify all the controls used for navigation and input. Are they all obvious? Discuss some differences in visibility and affordance among the controls.
3. A common maxim for designing a man-machine interface is that it is better to change the machine than to try to change the human to accommodate the machine. Are there machines (or systems) that you use in your daily life that have room for improvement? Are the current generations of Windows PC and Apple Mac as usable as they might be? If not, what improvements can you suggest? Is the World Wide Web as usable as it might be? If not, what improvements can you suggest? Are we just beginning to see some breakthroughs in usability, or have most of the big improvements already been made?
4. Download and install App Inventor from MIT (<http://www.appinventor.org/> and <http://appinventor.mit.edu/explore/>). Use it to develop a prototype interface that implements the storyboard in Figure 8-18.
5. Evaluate the course registration system at your university. List the basic steps of a user’s dialogue with the system. What are some of the problems with the system from the perspective of ease of learning and ease of use? In what ways is the system inflexible? In what ways is needed information not available? Is too much information provided that distracts from the task at hand?
6. Evaluate the online catalog system at your university library. Write a dialogue that shows the interaction between the user and the system. Rewrite the dialogue to improve it. Create a storyboard to show how your design would look and feel.
7. Find a Web site with direct customer ordering. Browse through some product descriptions and note the design of the dialogue and the Web pages. What do you like and dislike about the design? Evaluate the Web site based on visibility and affordance. Does the site achieve an optimal balance between the number of page refreshes and the delay between page refreshes? Would your answer change if you were using a different computing device, a different network, or accessing the system at a different time of day?

CASE STUDY

Video Rental Service

Develop a storyboard that encompasses the following sample dialogue. The system supports rental and download of movies or television shows to a computer or digital video

SYSTEM: What would you like to do?
 USER: I'd like to rent a movie.
 SYSTEM: Okay. What is your e-mail address or account number?
 USER: My e-mail address nwells22@gmail.com.
 SYSTEM: Fine. You are Nancy Wells at 1122 Silicon Avenue. Correct?
 USER: Yes.
 SYSTEM: What movie do you want to rent?
 USER: *Titanic*.
 SYSTEM: I have two available movies named *Titanic*. The first was released in 1953 and stars Barbara Stanwyck and Clifton Webb. The second was released in 1997 and stars Leonardo DiCaprio and Kate Winslet. Which do you want to rent?

recording and playback device. A dialogue between the system and user might follow this pattern:

USER: Both.
 SYSTEM: Fine. That is *Titanic* (the 1953 release) with a rental charge of \$2.99 for seven days and *Titanic* (the 1997 release) with a rental charge of \$5.99 for seven days. Would you like to add more rentals?
 USER: That's all.
 SYSTEM: Okay. The total charge is \$8.98. Shall I charge that amount to your credit card file (a Visa with an account number ending in 0899)?
 USER: Yes.
 SYSTEM: Your movies are downloading to your computer now. They will expire at midnight seven days from now.
 USER: Okay. Thanks.

RUNNING CASE STUDIES

Community Board of Realtors®

One of the key use cases for the Multiple Listing Service system is *Create new listing*, where the realtor enters all the important information about a new listing he or she has obtained. Realtors want to be able to create a new listing as soon as possible so other realtors and potential buyers will find the listing online. Some like to enter the information while talking with the owner or while inspecting the property. Realtors are rarely in their offices these days, so being able to create a new listing on a mobile device is a key feature of the Multiple Listing Service system.

Consider the information that must be entered when creating a new listing, and list the dialogue steps that are necessary. Keep in mind that when designing for a smartphone, less information can be entered in each step compared with a full screen Web application.

Also keep in mind that typing is error-prone and awkward for many users, so think about opportunities to use check boxes, radio buttons, and list boxes to aid selection. Create a storyboard of this use case for a mobile device, showing each step of the dialogue that maximizes the use of check boxes, radio buttons, and list boxes.

On the Spot Courier Services

Review the case description and your solution for the Web scenario of the use case *Request package pickup* from Chapter 5. Then, using a presentation tool, such as Microsoft PowerPoint or Apple Keynote, create a storyboard of the Web pages necessary to support the use case.

The case description in Chapter 5 also identified a new use case, which we can call *View scheduled pickups/deliveries*. Based on current technology, write a dialogue showing how this might be supported with a portable digital device such as an Internet-connected tablet. You may use any current technology that you

deem applicable, such as GPS tracking, map and directions software, and real-time updates of pickup locations. Consider the possibility that the driver may

want to get an overview of his or her stops for the entire run, view the next few stops, or just get directions to the next stop.

The Spring Breaks ‘R’ Us Travel Service

The Spring Breaks ‘R’ Us social networking subsystem requires an intuitive and engaging user-interface design for mobile devices. But the social networking subsystem can also play an important role in resort security. For example, each resort could use traveler location, interests, activities, and “likes”—all of which are available through the application—to monitor the well-being of travelers staying at the resort. Most spring break travelers are young, and their parents are concerned about their safety—particularly at isolated resorts in foreign countries. SBRU and the participating resorts could keep track of where travelers are and who they are near, monitor messages about activities and parties, and anticipate crowded conditions or vulnerable travelers wandering off-site. Alerts could notify security if conditions veer away from normal or if messages indicate there are problems. For example, if the pool is overcrowded, some action can be taken. If messages refer to places off-site that are known

to be dangerous, security can make an extra patrol. Although many people find this use of private information objectionable, others—particularly parents—find it essential.

Imagine resort security with a large, wide-screen monitor tracking traveler activities. Design a main screen that includes multiple locations, paths and roads, traveler location and status, messages traveling from traveler to traveler, and other features that security should monitor. Create a storyboard that shows an example of a pop-up alert and a menu of options that security might select after an alert. Should you also show security the staff members’ locations and status? How about clicking security staff members to send them a message? How about clicking a location to turn up the lights or to close a security gate? Be creative as you think through the design possibilities. You should include four or five screen layouts for the storyboard.

Sandia Medical Devices Real-Time Glucose Monitoring

In Chapter 3, you learned about a use case where the patients would like to view their history. In Chapter 5, we called the use case *View history*. Here are those paragraphs:

- Viewing and interpreting data and trends. Patients want to be able to view more than their current glucose level. They would like the ability to see past glucose levels over various time periods, with a specific focus on time periods during which their glucose was inside and outside of acceptable ranges. A graphical view of the data is preferred, although some patients also want to be able to see actual numbers.
- Entering additional data. Some patients want to be able to enter text notes or voice messages to supplement glucose level data. For example, patients who see a high glucose alert might record voice messages describing how they feel or what they had recently eaten. Some patients thought that sharing such information with their health-care providers might be valuable, but others only wanted such information for themselves.

You have been asked to write a smartphone app to display this history. The smartphone app would connect to the monitor using Bluetooth and collect the previous 24 hours’ data. Using a drawing program such

as Visio or PowerPoint, sketch out the screen to view this history. Explain your screen(s) and controls—how they work and what they mean. Remember the principles you have learned, especially about affordance and visibility, and the capabilities of smartphones.

Consider the following points:

1. How would you illustrate the graph? How could you use colors to indicate good/warning/critical readings?
2. How would you combine graphical display with actual numeric values? How would you display this information? What icons would you use?
3. How would you allow messages to be entered? Voice or text? How would you activate that feature? How would you play back or redisplay the messages?
4. How would you indicate the ability to transmit this data to the medical staff?
5. What changes would you make to the user interface if this were a tablet app instead of a smartphone? Sketch out the screen(s) for a tablet app.

The following Web site shows a very small “on the belt” display. Your smartphone will have more built-in capabilities, but this Web site can give you a starting point. <http://www.medtronicdiabetes.com/customer-support/device-settings-and-features/sensor-settings/read-sensor-graphs>

FURTHER RESOURCES

- Randolph G. Bias and Deborah J. Mayhew, *Cost-Justifying Usability: An Update for the Internet Age* (2nd ed.). Morgan Kaufmann, 2005.
- Paul C. Brown, *Implementing SOA: Total Architecture in Practice*. Addison-Wesley, 2008.
- Patrick Carey, *New Perspectives on Creating Web Pages with HTML, XHTML, and XML* (3rd ed.). Cengage Learning, 2010.
- Jakob Nielsen and Raluca Budi, *Mobile Usability*, New Riders, 2013.
- Donald Norman, *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books, 2013.
- Janice Redish, *Letting Go of the Words: Writing Web Content that Works*. Morgan Kaufmann, 2007.
- Ben Shneiderman, Catherine Plaisant, Maxine Cohen, and Steven Jacobs, *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (5th ed.). Addison-Wesley, 2009.
- Joel Sklar, *Principles of Web Design* (5th ed.). Cengage Learning, 2012.



Designing the Database

CHAPTER NINE

CHAPTER OUTLINE

- Databases and Database Management Systems
- Database Design and Administration
- Relational Databases
- Distributed Database Architectures
- Protecting the Database

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- Explain the responsibilities of the data administrator and database administrator
- Design a relational database schema based on a class diagram
- Evaluate and improve the quality of a database schema
- Describe the different methods for configuring distributed databases
- Explain the importance of and methods for protecting the database

OPENING CASE DOWNSLOPE SKI COMPANY: A NEW PRODUCTION DATABASE

Downslope Ski Company manufactures skis and snowboards. In the company's early years, ski manufacturing was simple and straightforward. However, in recent years, manufacturing has become more complex with high-tech materials, such as carbon-laced resins, and product features, such as skis with integrated bindings. To ensure product quality, computers now control production processes, including ingredient mixtures, furnace temperatures, and curing times. These changes have gone hand in hand with increased attention to raw materials quality.

Downslope uses a just-in-time (JIT) manufacturing process, which means that it doesn't stockpile a large quantity of raw materials. It keeps about a 10-day supply on hand and depends on its suppliers to restock materials at least weekly. As with many growing companies, their software systems have lagged behind changes in the manufacturing process. Downslope is currently updating its manufacturing control system. Critical issues in the new system include the database and accessibility of its information to all the interested parties.

Nathan Jones wants to get a good grasp of the overall database requirements before proceeding with the

design for the first iteration development. There are several areas he wants to review, including data sources, destinations, and formats; data requirements for every step in the manufacturing process; requirements for historical data; data size and volume; database response time requirements; and so forth.

Input data sources and formats are especially important because suppliers need access to inventory levels, including both items that are in production and supplies that are on order. They also need to update the database for deliveries and billing. Other priorities include capturing and storing data from production furnaces and manufacturing equipment, including periodic outputs of volumes, temperatures, and times. Of course, production staff need access to the database to control production and to forecast future needs. Finally, sales staff are interested in product levels and delivery dates for finished inventory. The inputs and outputs range from Web-based access, to automated equipment feeds, to internal user access. These very diverse requirements must be integrated into a comprehensive database and information system.

■ Overview

Databases and database management systems are core components of modern information systems. Almost every facet of our lives depends on databases. Databases underlie everyday activities, including making a phone call, searching the Web, reading your e-mail, buying groceries, visiting your doctor, and tweeting your favorite friends. Database management systems let database designers, developers, and end users create databases and store, retrieve, and manage the data. Sharing and managing the vast amounts of data needed by a modern organization wouldn't be possible without a database management system.

In Chapter 5, you learned to construct a domain model class diagram. The class diagram is the starting point for database development. To design and build an information system, developers must transform conceptual models into more detailed design models. In this chapter, you will learn how to transform the domain model class diagram into a detailed database model and how to implement that model with a database management system.

■ Databases and Database Management Systems

database (DB) an integrated collection of stored data that is centrally managed and controlled

database management system (DBMS) a system software component that manages and controls one or more databases

A **database (DB)** is an integrated collection of stored data that is centrally managed and controlled. A database typically stores information about dozens or hundreds of classes or entities. A database is managed and controlled by a **database management system (DBMS)**. A DBMS is a system software component that is generally purchased and installed separately from other system software components (e.g., operating systems). Examples of modern database management systems include Microsoft SQL Server™, Oracle, and MySQL™.

FIGURE 9-1 Database and DBMS components with interacting actors

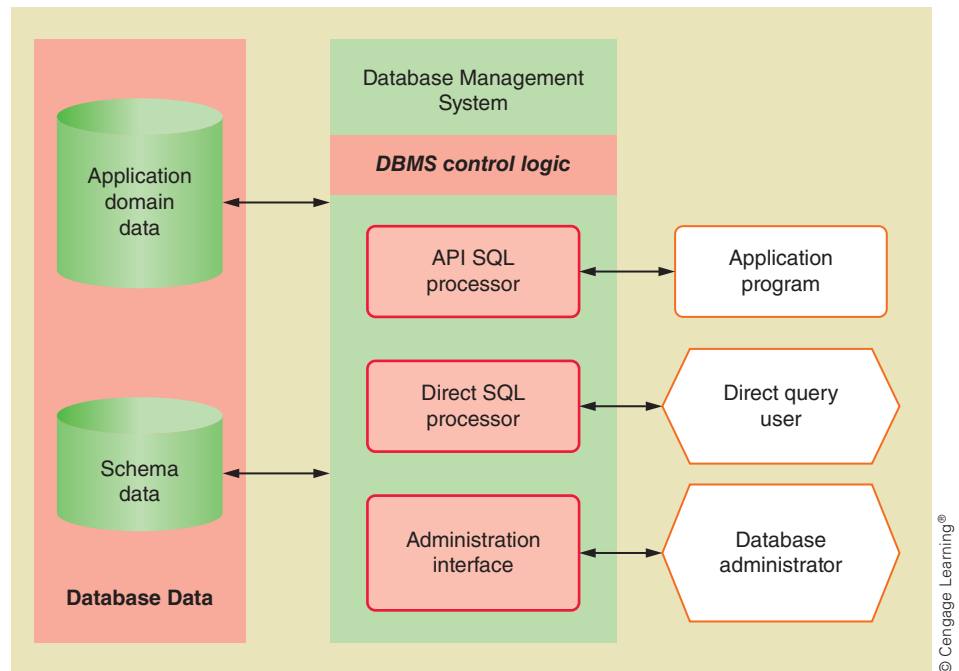


Figure 9-1 illustrates the components of a typical database. Usually when we refer to a database, we mean both the database data, shown on the left, and the DBMS, shown in the middle. The database data consists of two related information stores: the application domain data, which is the actual data for the problem domain, and the **schema**, which contains descriptive information about the application domain data. The reason we usually include all of these components when we discuss the database is that the application domain data cannot be accessed without the DBMS and the schema data. Hence, we consider it all together.

The schema data is required because it describes the structure and rules for accessing the application domain data. It is often referred to as *metadata*, or data about the data. It includes the following:

- Organization of individual stored data items into higher-level groups, such as tables
- Associations among tables (e.g., pointers from customer objects to related sale objects)
- Details of individual data items, including types, lengths, locations, and indexing of data items
- Access and content controls, including allowable values for specific data items, value dependencies among multiple data items, and lists of users allowed to read or update data items

A DBMS has four key components: an application programming interface (API), a direct SQL or query interface, an administrative interface, and an underlying set of data-access programs and subroutines. Figure 9-1 refers to *SQL*, which stands for **Structured Query Language SQL**, as the query language used to access the database data. Application programs, users, and administrators never access the domain data directly. Instead, they tell an appropriate DBMS interface what data they need to read or write, using names defined in the schema. The DBMS accesses the schema to verify that the requested data exist and that the requesting user has appropriate access privileges. If the request is valid, the DBMS extracts information about the physical organization of the requested data from the schema and uses that information to access the domain data on behalf of the requesting program or user.

schema database component that contains descriptive information about the data stored in the physical data store

Structured Query Language SQL a query language used to access and update the data in a relational database

Databases and database management systems provide several important data access and management capabilities, including the following:

- Simultaneous access by many users and application programs
- Access to data without writing application programs (i.e., via the Direct SQL processor)
- Application of uniform and consistent access and content controls
- Integration of data stored on multiple servers distributed across multiple locations

DBMSs have evolved through a number of technology stages since their introduction in the 1960s. The most significant change has been the type of model used to represent and access the content of the physical data store. Early models, including the hierarchical and network models, have been replaced by the relational and object-oriented (OO) models. However, though most modern software is designed and implemented by using OO techniques and programming languages, most deployed databases and DBMSs are based on the relational model.

■ Database Design and Administration

Before jumping into the actual details of how to design the database, you first need to address three important issues:

- How does database design integrate into the organization's overall technological environment?
- How does database design integrate into the overall project plan?
- Who is involved in database design?

■ Technology Environment

Most system development occurs within an organization that has an existing technology architecture and upgrade strategy. Included in the technology architecture may be various DBMSs, databases, database servers, and networks. The following sections first consider the software and data environment, and then discuss the hardware and server configurations.

■ Data and DBMS Environment

In most organizations, information systems and databases are deployed in a piecemeal fashion, with new systems purchased or developed to meet specific business requirements. The result, of course, is many independent systems with diverse data and DBMS configurations in one company. In addition, one of the most important assets of almost any organization is its data. Information such as client lists, product descriptions, and sales histories is critically important to the continued success of the organization. The development of a new information system, and especially the database design, must capture and preserve this critical information. In other words, the new database must incorporate the important data from the previous databases and systems. (Chapter 13 discusses issues related to the actual conversion of the data from the old database to the new database.)

In addition to replacing or upgrading existing data in the database itself, it is also common that the previous system also fed data to other information systems in the organization. This same service will need to be reflected in the design of the new database. For example, a new inventory system may need to be tightly coupled to a supply chain management system with data flowing in both directions between the two databases.

These issues become even more important, and more complex, when different databases are residing on different DBMSs. Data transfer between DBMSs is

more complex. Each DBMS has its own rules and idiosyncrasies with regard to data types, SQL extensions, user authentication and privileges, triggers, stored procedures, and so forth.

■ Hardware and Network Architecture

The other big issue is the hardware and network configuration in the organization. An important issue is how the database architecture of the new system will interact with the architecture that supports existing systems. More specific questions include whether the new system will use existing DBMSs and servers—it is possible for a single DBMS or server to support multiple independent databases—and whether the existing servers and the network have sufficient unused capacity to support the new system. Those questions are typically asked and answered in an early project iteration or before the project is even approved.

A beginning organization may only have a few desktop systems. However, even small organizations can have a local area network (LAN), which connects desktop systems together and permits sharing of database information. Frequently, the need for a new system and database is due to growth or anticipated growth. In those cases, the network and computer server environment is changing and growing. Decisions about the deployment architecture should be made early in the project and will have a big impact on database design decisions. Going from smallest to largest hardware architecture, databases can be deployed as the following:

- A single desktop system residing on one computer and used only by one or two users.
- A shared database that resides on a database server for a LAN. A database of this size can be accessed by all the users in a particular office or department. A single server database might also be deployed over the Internet if it is accessed by a Web-based application.
- A larger database requiring multiple servers, but that is contained within a single server farm or data center. This configuration requires a sophisticated switchboard function with decision and routing protocols to respond to queries and updates.
- Finally, when the organization becomes global with many users worldwide (or in multiple locations in a single country), it is often required to have multiple data centers or server farms. As you can imagine, this requires incredible sophistication to maintain current data among the various data centers. Later in this chapter, you will learn about distributed databases that exist in this configuration.

■ Project Plan and Schedule

Choosing the correct timing for database design and construction within a system development project isn't a simple matter. The basic issue is whether the database design can proceed in iterations or if it is better to complete the database early in the project. In Chapter 1, you saw an example of an iterative approach to system development using Agile techniques. (You will learn more about iterative development in Chapters 10 and 11.) The philosophy of an iterative approach is to develop the system incrementally, including determining the processing requirements and mapping the classes of the domain model. Because the primary input for database design is the domain model, an incremental development of the domain model dictates an incremental design of the database. However, there exists a fundamental tension between Agile/iterative development and designing the complete database early. The foundation of any information system is the database, and having a stable database design facilitates the development of the entire system.

Iterative development and phased deployment typically occur together in current adaptive approaches to system development. The philosophy of using

iterations is that a subsystem will “grow” piecemeal as requirements are discovered. Phased deployment is normally implemented by deploying various subsystems one at a time until the entire system has been deployed. Database design is influenced by both of these techniques.

Normally in iterative development, the higher-risk elements of a subsystem are developed first. Because the database is the foundation of any information system, it is common to define the domain model and perform the database design in the first few iterations of subsystem development. For example, rather than focus on a detailed design of the user interface and on programming in the early iterations, it is usually a better approach to expend time and resources to discover and validate all the classes and attributes of the domain model.

Unfortunately, even this technique may not be sufficient for a complex system consisting of multiple subsystems with interlocking database requirements. Focusing only on a single subsystem within a set of iterations may not discover important database considerations influenced by the design of other subsystems not yet in development. In a phased deployment, these other subsystems may not be explored until the first subsystem has been deployed. Obviously, if major changes to the database are required, the original deployment may need to be redone. There are no definitive directions to determine if the entire database must be designed before any portion can be deployed. Experienced project managers and team members are sensitive to the risks associated with phased database development and deployment.

■ Database Design Team

Larger organizations typically have staff members dedicated to maintaining the operational databases that underlie an organization’s information systems. Although the project team has primary responsibility for defining the domain model and designing the corresponding database for a new system, permanent database staff usually assist the project team in database design. This ensures consistency between database requirements for new and existing systems and helps to avoid problems that might disrupt existing systems when a new system is deployed.

Two key positions on the permanent database staff are the data administrator and the database administrator. A **data administrator (DA)** has responsibility for the structure and integrity of the data itself. Specifically, the DA manages important aspects of data definition and database design, including the following:

- **Data standards.** Naming standards, definition standards, data typing standards, and value edits
- **Data use.** Ownership of data, accessibility of data, and confidentiality
- **Data quality.** Validation rules, completeness, currency, consistency, and relevancy

A **database administrator (DBA)** maintains the database after it has been deployed and manages the safety and operation of the database. It is his or her job to ensure that the database—both the data and the DBMS—is configured correctly for the organization’s architecture and performs effectively and efficiently. The DBA’s responsibilities include the following:

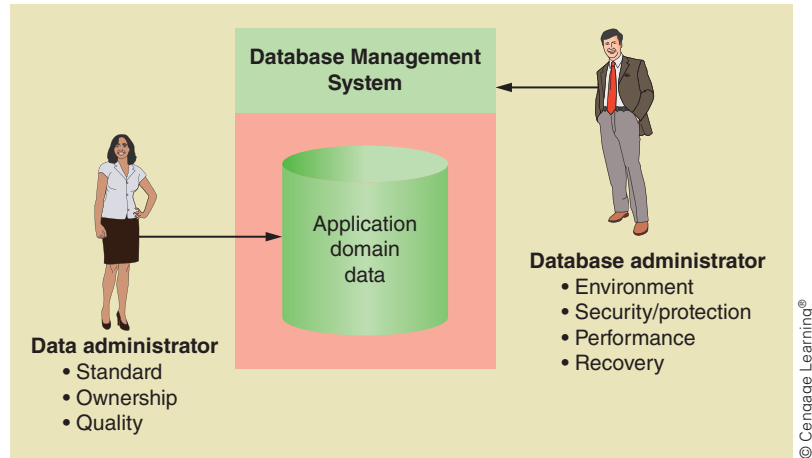
- Managing a multiple DBMS environment
- Protecting the data and database, including user authentication and attack prevention
- Monitoring and maintaining high levels of performance
- Backing up the database and defining recovery procedures

Figure 9-2 illustrates the different focus and responsibilities of the DA and the DBA.

data administrator (DA) The person in charge of the structure and integrity of the data

database administrator (DBA) The person in charge of the safety and operation of the DBMS

FIGURE 9-2 Data and database administrators' responsibilities



■ Relational Databases

Historically, many different structures have been used to organize data into a database. Most early structures were either hierarchical or linked together in some type of network. More recently, however, databases have been built in a relational structure by defining a set of interconnected tables. The following section defines the terms and concepts of relational databases.

■ Definitions

relational database management system (RDBMS) a DBMS that organizes data in tables or relations

tables two-dimensional data structures consisting of columns and rows

row one horizontal group of data attribute values in a table

attribute one vertical group of data attribute values in a table

attribute value the value held in a single table cell

A **relational database management system (RDBMS)** is a DBMS that organizes stored data into structures called **tables** or *relations*. Relational database tables are similar to conventional tables; that is, they are two-dimensional data structures of columns and rows. However, relational database terminology is somewhat different from conventional table and file terminology. A single row of a table is called a **row**, *tuple*, or *record*, and a column of a table is called an **attribute** or *field*. A single cell in a table is called an **attribute value**, *field value*, or *data element*. As indicated previously, the data in a relational database is edited and accessed through a common query language called Structured Query Language or SQL. You will learn how to use SQL in your database courses.

Figure 9-3 shows the content of a table as displayed by the Microsoft Access relational DBMS. Note that the first row of the table contains a list of attribute names (column headings) and that the remaining rows contain a collection of attribute values, each of which describes a specific product. Each row contains the same attributes in the same order, but not necessarily containing the same attribute values.

FIGURE 9-3 Partial display of a relational database table

The screenshot shows a table with the following data:

ProductItemID	Gender	Description	Supplier	Manufacturer	Picture
10564	Both	Super Alpine Performance Skis	K2	K2	
10766	Man	Extreme Ski Boots	Nordica	Nordica	
1244	Man	Casual Chino Trousers	West Coast	Adida	
1245	Man	Fleece Crew Sweatshirt	West Coast	Adida	
1246	Man	Fleece Crew Sweatshirt V-Neck	West Coast	Adida	
1247	Man	Fleece Crew Sweatshirt Zippered	West Coast	Adida	
1248	Man	Solid Color Flannel Shirt	RMO	RMO	
1249	Man	Plaid Flannel Shirt	RMO	RMO	
1250	Man	Polo Shirt	RMO	RMO	
1251	Man	Polo Shirt Zippered	RMO	RMO	
1252	Man	Navigator Jacket	Colorado Supply	North Face	
1253	Man	Navigator Jacket Hooded	Colorado Supply	North Face	
1254	Man	Cotton Thermal Shirt	Colorado Supply	Under Armour	

key an attribute or set of attributes, the values of which are unique for each row of the table, that is used to uniquely identify a row

candidate key an attribute or set of attributes that are unique identifiers and could serve as the primary key

primary key the key chosen by a database designer to define relationships by being used as a foreign key in other tables

foreign key an attribute that duplicates the primary key of a different (or foreign) table

Each table in a relational database must have a unique key. A **key** is an attribute or set of attributes, the values of which occur only once in all the rows of the table. You learned about a key in Chapter 4 when you learned about object classes. The key uniquely identifies each object in a class. For a relational database table, the key uniquely identifies each row in the table. Frequently, there may be multiple groups of attributes that are unique in each row and that could serve as a **candidate key**. If there are multiple unique attributes or sets of attributes that are candidate keys, then the database designer must choose one of the possible keys as the **primary key**. In any event, each table is given a primary key.

Key attributes may be natural or invented. An example of a natural key attribute in chemistry is the atomic weight of an element (a unique identifying characteristic) in a table containing descriptive data about elements. Unfortunately, in business, few natural key attributes are useful for information processing, so most key attributes in a relational database are invented. Your wallet contains many examples of invented keys, including your Social Security number, driver's license number, and credit card numbers. The key field is an important element because it is frequently used to find or locate a particular row in a table. For example, in a driver's license system, using your driver's license number, your record can be retrieved with information about you and your driving history.

In addition to serving as unique identifiers for each row, primary keys are critical elements of relational database design because they are the basis for representing relationships among tables. Keys are the “glue” that binds rows of one table to rows of another table—in other words, keys relate tables to each other. For example, consider the class diagram fragment from the RMO example, which is shown in **Figure 9-4**, and the tables that are shown in **Figure 9-5**. The class diagram fragment shows an optional one-to-many association between the classes `ProductItem` and `InventoryItem`. The upper table in Figure 9-5 contains data representing the `ProductItem` class. The lower table contains data representing the `InventoryItem` class.

The association between the `ProductItem` and `InventoryItem` classes is represented by a common attribute value within their respective tables. The `ProductItemID` attribute (the primary key of the `ProductItem` table) is stored within the `InventoryItem` table, where it is called a foreign key. A **foreign key** is an attribute that duplicates the primary key of a different (or foreign) table. In Figure 9-4, the existence of the value 1244 as a foreign key within the `InventoryItem` table indicates that the values of `Vendor`, `Gender`, and `Description` in the row defined by the value 1244 of the `ProductItem` table also describe inventory items 86779 through 86788.

■ Designing Relational Databases

For database design, the preferred starting point is the domain model class diagram because it provides a comprehensive description of all the

FIGURE 9-4 Portion of the RMO class diagram

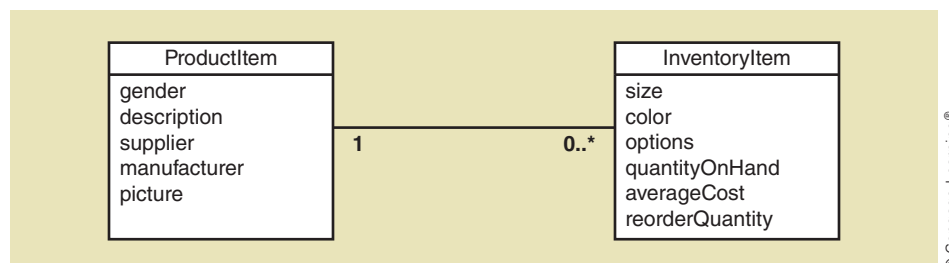


FIGURE 9-5 An association between rows in two tables represented by primary and foreign keys

ProductItem						
ProductItemID	Gender	Description	Supplier	Manufacturer	Picture	
10564	Both	Super Alpine Performance Skis	K2	K2		
10766	Man	Extreme Ski Boots	Nordica			
1244	Man	Casual Chino Trousers				
1245	Man	Fleece Crew Sweatshirt				
1246	Man	Fleece Crew Sweatshirt V-Neck				
1247	Man	Fleece Crew Sweatshirt Zippered				
1248	Man	Solid Color Flannel Shirt				
1249	Man	Plaid Flannel Shirt				
1250	Man	Polo Shirt				
1251	Man	Polo Shirt Zippered				

InventoryItem							
InventoryItemID	ProductItemID	Size	Color	Options	QuantityOnHand	Average Cost	RecorderQuantity
86779	1244	30/30	Khaki		45	\$12.75	100
86780	1244	30/30	Slate		10	\$12.75	100
86781	1244	30/30	LightTan		17	\$12.75	100
86782	1244	30/31	Khaki		22	\$12.75	100
86783	1244	30/31	Slate		6	\$12.75	100
86784	1244	30/31	LightTan		31	\$12.75	100
86785	1244	30/32	Khaki		120	\$12.75	100
86786	1244	30/32	Slate		28	\$12.75	100
86787	1244	30/32	LightTan		21	\$12.75	100
86788	1244	30/33	Khaki		7	\$12.75	100

Source: Microsoft Corporation

problem domain classes. To create a relational database schema from a domain model class diagram, follow these steps:

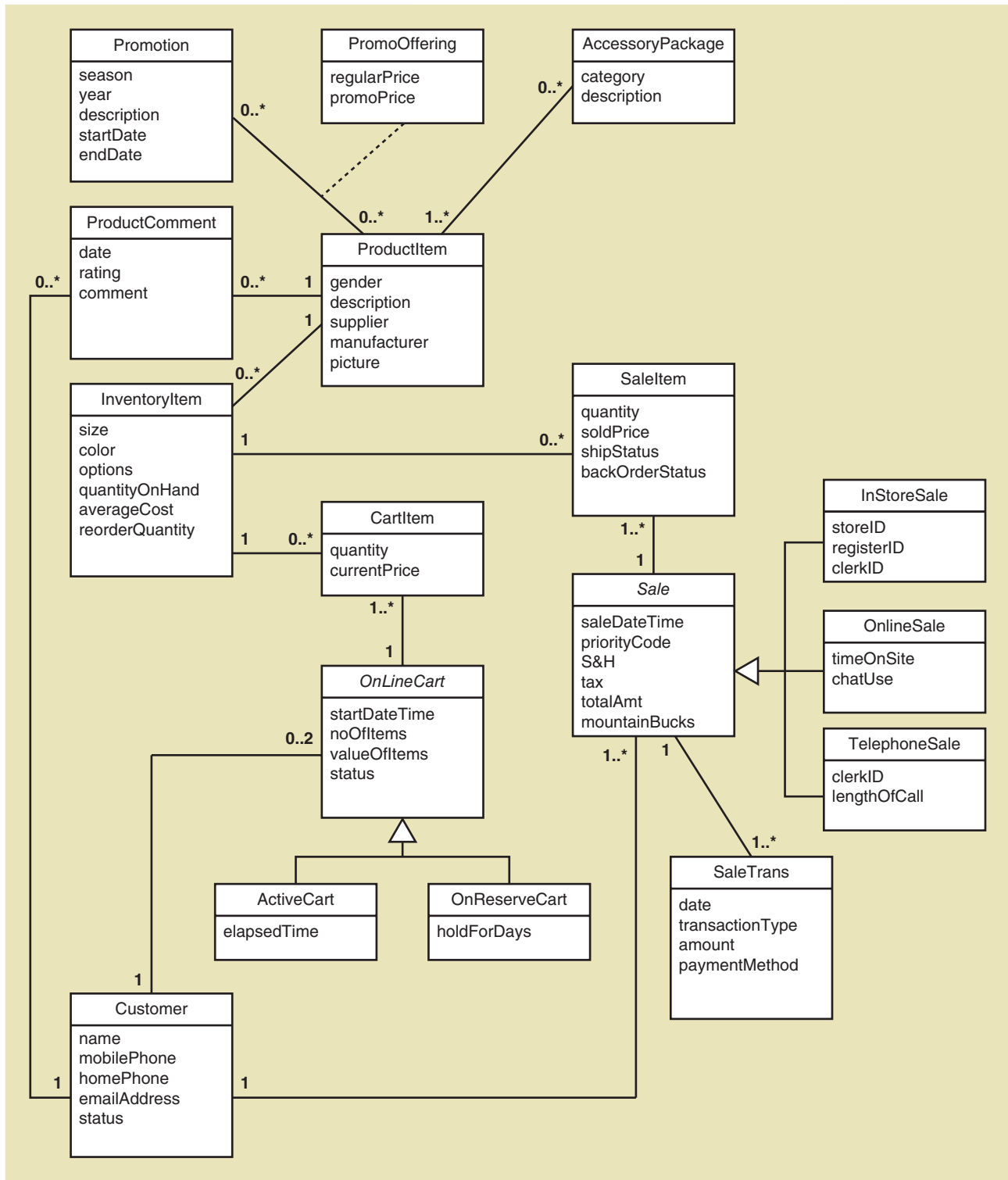
1. Create a table for each class in the domain model.
2. Choose a primary key for each table (invent one, if necessary).
3. Add foreign keys to represent one-to-many associations.
4. Create new tables to represent many-to-many associations.
5. Represent and adjust classification hierarchies.
6. Define referential integrity constraints.
7. Evaluate schema quality and make necessary improvements.
8. Choose appropriate data types.
9. Incorporate integrity and security controls.

The following subsections discuss these items in more detail.

■ Creating Tables from Domain Classes

The first step in creating a relational database schema is to create a table for each class on the class diagram. **Figure 9-6** shows a partial class diagram for the RMO customer support system with 17 classes, including three specialized classes of Sale and two of OnlineCart. For the moment, we will treat each group of generalized and specialized classes as if it were a single class. Thus, there are 12 primary classes that will be used to create tables. The attributes of each table will be the same as those defined for the corresponding class in the class diagram. To avoid confusion, table and attribute names should match the names used in the class diagram and abbreviations should be avoided. Initial table definitions for the classes in **Figure 9-6** are shown in **Figure 9-7**. This example uses the same camel case naming convention that was used to name the attributes in the class diagram. However, there are other conventions that are popular in industry. Some DBMSs allow attribute names with embedded spaces. Other conventions include using all lowercase letters with underscores. The examples in this book use lowercase attribute names in the domain model and capitalized column names in the database schema.

FIGURE 9-6 Subset of the RMO domain model class diagram



© Cengage Learning®

■ Choosing Primary Keys

After creating tables for each class, the designer selects a primary key for each table. If a table already has an attribute or set of attributes that are guaranteed to be unique, then the designer can choose that attribute or set of attributes as the primary key. If the table contains no possible keys, then the designer must

FIGURE 9-7 Initial set of tables representing RMO classes

Table	Attributes
AccessoryPackage	Category, Description
CartItem	Quantity, CurrentPrice
Customer	Name, MobilePhone, HomePhone, EmailAddress, Status
InventoryItem	Size, Color, Options, QuantityOnHand, AverageCost, ReorderQuantity
OnlineCart	StartDateTime, NumberOfItems, ValueOfItems, Status, ElapsedTime, HoldForDays
ProductComment	Date, Rating, Comment
ProductItem	Gender, Description, Supplier, Manufacturer, Picture
PromoOffering	RegularPrice, PromoPrice
Promotion	Season, Year, Description, StartDate, EndDate
Sale	SaleDateTime, PriorityCode, ShippingAndHandling, Tax, TotalAmount, MountainBucks, StoreID, RegisterID, ClerkID, TimeOnSite, ChatUse, LengthOfCall
SaleItem	Quantity, SoldPrice, ShipStatus, BackOrderStatus
SaleTransaction	Date, TransactionType, Amount, PaymentMethod

© Cengage Learning®

FIGURE 9-8 Class tables with primary keys identified in bold

Table	Attributes
AccessoryPackage	AccessoryPackageID , AccessoryCategory, Description
CartItem	CartItemID , Quantity, CurrentPrice
Customer	AccountNumber , Name, MobilePhone, HomePhone, EmailAddress, Status
InventoryItem	InventoryItemID , Size, Color, Options, QuantityOnHand, AverageCost, ReorderQuantity
OnlineCart	OnlineCartID , StartDateTime, NumberOfItems, ValueOfItems, Status, ElapsedTime, HoldForDays
ProductComment	ProductCommentID , Date, Rating, Comment
ProductItem	ProductItemID , Gender, Description, Supplier, Manufacturer, Picture
PromoOffering	PromoOfferingID , RegularPrice, PromoPrice
Promotion	PromotionID , Season, Year, Description, StartDate, EndDate
Sale	SaleID , SaleDateTime, PriorityCode, ShippingAndHandling, Tax, TotalAmount, MountainBucks, StoreID, RegisterID, ClerkID, TimeOnSite, ChatUse, LengthOfCall
SaleItem	SaleItemID , Quantity, SoldPrice, ShipStatus, BackOrderStatus
SaleTransaction	SaleTransactionID , Date, TransactionType, Amount, PaymentMethod

© Cengage Learning®

invent a new key attribute. Any name can be chosen for an invented key field, but the name should indicate that the attribute contains unique values. Typical names include Code, Number, and ID—possibly combined with the table name (e.g., ProductCode and OrderID). **Figure 9-8** shows the class tables with primary key columns in boldface type.

Because key creation and management are critical functions in databases and information systems, many relational DBMSs automatically create a primary key. Furthermore, DBMSs typically provide a special data type for invented keys

(e.g., the AutoNumber type in Microsoft Access). The DBMS automatically assigns a key value to newly created rows. The application program must then retrieve that key for use in subsequent database operations. Embedding this capability in the DBMS frees the Information Systems (IS) developer from designing and implementing customized key-creation software modules. In the example of Figure 9-8, all the primary keys will be an AutoNumber type of field except the AccountNumber for the Customer table. An AccountNumber may be an AutoNumber field, but it might just as easily be a specific number assigned, based on some other algorithm.

Invented keys that aren't assigned by the information system or DBMS must be carefully scrutinized to ascertain their uniqueness and usefulness over time. For example, employee databases in the United States commonly use Social Security numbers as keys. Because the U.S. government has a strong interest in guaranteeing the uniqueness of Social Security numbers, the assumption that they will always be unique seems safe. But will all employees who are stored in the database have a Social Security number? What if the company opens a manufacturing facility in Asia or South America?

Invented keys assigned by nongovernmental agencies deserve even more careful scrutiny. For example, FedEx, UPS, and other shipping companies assign a tracking number to each shipment they process. Tracking numbers are guaranteed to be unique at any given point in time, but are they guaranteed to be unique forever (i.e., are they ever reused)? Could reuse of a tracking number cause primary key duplication in your database? And what would happen if two different shippers assigned the same tracking number to two different shipments?

Organizations that have large number of records in the database and that share these items across multiple databases will sometimes use a globally unique identifier (GUID). GUIDs are guaranteed to be unique across all items in the world. However, because they are usually 32 hexadecimal digits in length, they require substantial storage space in the database, and are usually only used for specific items that need to be uniquely identified outside of the database.

■ Representing Associations

Associations are represented within a relational database by foreign keys. The choice of which foreign keys to place in which tables depends on the type of association being represented. The RMO class diagram in Figure 9-6 contains 10 one-to-many associations, two many-to-many associations, and two generalization/specialization association groups. You will deal with the generalization/specialization associations in a later step. The rules for representing one-to-many and many-to-many associations are as follows:

- **One-to-many associations.** Add the primary key attribute(s) of the “one” class to the table that represents the “many” class.
- **Many-to-many associations.** There are two possible situations—either an association class has already been defined in the domain model and a table has been created, or a many-to-many association exists with no association class defined:
 - **Association with defined association class.** Add the primary keys of the endpoint classes (i.e., the associated classes) as foreign keys in the table that was created for the association class. These foreign keys are always a candidate key and frequently will serve as the primary key. They uniquely define the rows in the association class table.
 - **Association without a class.** Create a new table to represent the association. Add the primary keys of the endpoint classes (i.e., the associated classes) as foreign keys in the new table. These foreign keys are always a candidate key. The concatenation of these primary keys is always a unique identifier for the record and thus a candidate key. An invented primary key may also be added if desired for ease of use.

FIGURE 9-9 One-to-many associations represented by adding foreign key attributes (shown in italic)

Table	Attributes
AccessoryPackage	AccessoryPackageID , AccessoryCategory, Description
CartItem	CartItemID , <i>InventoryItemID</i> , <i>OnlineCartID</i> , Quantity, CurrentPrice
Customer	AccountNumber , Name, MobilePhone, HomePhone, EmailAddress, Status
InventoryItem	InventoryItemID , <i>ProductItemID</i> , Size, Color, Options, QuantityOnHand, AverageCost, ReorderQuantity
OnlineCart	OnlineCartID , <i>CustomerAccountID</i> , StartDateTime, NumberOfItems, ValueOfItems, Status, ElapsedTime, HoldForDays
ProductComment	ProductCommentID , <i>ProductItemID</i> , <i>CustomerAccountNumber</i> , Date, Rating, Comment
ProductItem	ProductItemID , Gender, Description, Supplier, Manufacturer, Picture
PromoOffering	PromoOfferingID , RegularPrice, PromoPrice
Promotion	PromotionID , Season, Year, Description, StartDate, EndDate
Sale	SaleID , <i>CustomerAccountNumber</i> , SaleDateTime, PriorityCode, ShippingAndHandling, Tax, TotalAmount, MountainBucks, StoreID, RegisterID, ClerkID, TimeOnSite, ChatUse, LengthOfCall
SaleItem	SaleItemID , <i>InventoryItemID</i> , <i>SaleID</i> , Quantity, SoldPrice, ShipStatus, BackOrderStatus
SaleTransaction	SaleTransactionID , <i>SaleID</i> , Date, TransactionType, Amount, PaymentMethod

© Cengage Learning®

One-to-Many Associations Figure 9-9 shows the results of representing the 10 one-to-many associations within the tables from Figure 9-8. Each foreign key (shown in italic) represents a single association between the table containing the foreign key and the table that uses that same key as its primary key. For example, the attribute *CustomerAccountNumber* was added to the Sale table as a foreign key representing the one-to-many association between the Customer and Sale classes. The foreign key *SaleID* was added to the SaleTransaction table to represent the one-to-many association between Sale and SaleTransaction.

Many-to-Many Associations Figure 9-10 expands the table definitions in Figure 9-9 by updating the PromoOffering table to represent the many-to-many association between Promotion and ProductItem. The primary key of PromoOffering becomes the combination of *PromotionID* and *ProductItemID*. The old primary key *PromoOfferingID* is discarded. The two attributes that make up the primary key are also foreign keys and are displayed in boldface and italic to indicate their dual status. *PromotionID* is a foreign key from the Promotion table, and *ProductItemID* is a foreign key from the ProductItem table.

Because there is no association class representing the many-to-many association between ProductItem and AccessoryPackage, a new table named AccessoryPackageContents is created. As with PromoOffering, it contains two foreign key columns that combine to form the primary key.

Notice that the class diagram contains other classes—including ProductComment, SaleItem, and CartItem—that are connected between two other classes on the many side of one-to-many associations. Though not shown the same way as PromoOffering on the class diagram, SaleItem and CartItem are similar to PromoOffering in that they could also have been modeled as an association class in a many-to-many association. For example, SaleItem can also be thought of as a many-to-many association between Sale and InventoryItem.

FIGURE 9-10 *PromoOffering* table modified to represent the many-to-many association between *Product* and *Promotion*

Table	Attributes
AccessoryPackage	AccessoryPackageID , AccessoryCategory, Description
AccessoryPackageContents	AccessoryPackageID , ProductItemID
CartItem	InventoryItemID , OnlineCartID , Quantity, CurrentPrice
Customer	AccountNumber , Name, MobilePhone, HomePhone, EmailAddress, Status
InventoryItem	InventoryItemID , ProductItemID , Size, Color, Options, QuantityOnHand, AverageCost, ReorderQuantity
OnlineCart	OnlineCartID , <i>CustomerAccountID</i> , StartDateTime, NumberOfItems, ValueOfItems, Status, ElapsedTime, HoldForDays
ProductComment	ProductCommentID , <i>ProductItemID</i> , <i>CustomerAccountNumber</i> , Date, Rating, Comment
ProductItem	ProductItemID , Gender, Description, Supplier, Manufacturer, Picture
PromoOffering	PromotionID , ProductItemID , RegularPrice, PromoPrice
Promotion	PromotionID , Season, Year, Description, StartDate, EndDate
Sale	SaleID , <i>CustomerAccountNumber</i> , SaleDateTime, PriorityCode, ShippingAndHandling, Tax, TotalAmount, MountainBucks, StoreID, RegisterID, ClerkID, TimeOnSite, ChatUse, LengthOfCall
SaleItem	InventoryItemID , SaleID , Quantity, SoldPrice, ShipStatus, BackOrderStatus
SaleTransaction	SaleTransactionID , <i>SaleID</i> , Date, TransactionType, Amount, PaymentMethod

© Cengage Learning®

Because the combination of foreign key values in *SaleItem* is always unique, the foreign key combination can serve as the table's primary key, and the invented key (*SaleItemID*) created earlier can be discarded. A similar situation exists for *CartItem*. However, *ProductComment* is different. Even though it has one-to-many associations with *ProductItem* and *Customer*, it cannot be modeled as an association class. With *ProductComment*, it is possible for a single customer to make multiple comments about the same product. Hence, it is not an association class. If we tried to model it with only foreign keys as the primary key there would be two rows in the *ProductComment* table with the same values of *CustomerAccountNumber* and *ProductItemID*. Thus, the invented key *ProductCommentID* is retained and the two foreign key attributes aren't part of the primary key.

■ Representing Classification Hierarchies

Generalization/specialization hierarchies, such as the associations among *Sale*, *InStoreSale*, *TelephoneSale*, and *WebSale*, are a special case in relational database design. Just as a specialized class inherits the data and methods of a generalized class, a table representing a specialized class inherits some or all of its data from the table representing its generalized class. This inheritance can be represented in multiple ways, including the following:

- Combining all the tables into a single table containing the superset of all classes
- Using separate tables to represent the child classes, and using the primary key of the parent class table as the primary key of the child class tables
- Some combination of the previous two methods

FIGURE 9-11 Specialization classes of Sale and OnlineCart included in the primary tables

Table	Attributes
AccessoryPackage	AccessoryPackageID , AccessoryCategory, Description
AccessoryPackageContents	AccessoryPackageID , ProductItemID
CartItem	InventoryItemID , OnlineCartID , Quantity, CurrentPrice
Customer	AccountNumber , Name, MobilePhone, HomePhone, EmailAddress, Status
InventoryItem	InventoryItemID , ProductItemID , Size, Color, Options, QuantityOnHand, AverageCost, ReorderQuantity
OnlineCart	OnlineCartID , CustomerAccountID , StartDateTime, NumberOfItems, ValueOfItems, Status, ElapsedTime, HoldForDays
ProductComment	ProductCommentID , ProductItemID , CustomerAccountNumber , Date, Rating, Comment
ProductItem	ProductItemID , Gender, Description, Supplier, Manufacturer, Picture
PromoOffering	PromotionID , ProductItemID , RegularPrice, PromoPrice
Promotion	PromotionID , Season, Year, Description, StartDate, EndDate
Sale	SaleID , CustomerAccountNumber , SaleDateTime, PriorityCode, ShippingAndHandling, Tax, TotalAmount, MountainBucks, StoreID, RegisterID, ClerkID, TimeOnSite, ChatUse, LengthOfCall
SaleItem	InventoryItemID , SaleID , Quantity, SoldPrice, ShipStatus, BackOrderStatus
SaleTransaction	SaleTransactionID , SaleID , Date, TransactionType, Amount, PaymentMethod

© Cengage Learning®

Any of the three methods are acceptable approaches to representing a classification hierarchy.

Figure 9-11 shows the definition of the Sale table under the first method. All the non-key attributes in the InStoreSale, TelephoneSale, and OnlineSale classes are stored in the Sale table. For any particular sale, some of the attribute values in each row will be empty or, in database terminology, NULL. For example, a row representing a telephone sale would have no values for the attributes StoreID, RegisterID, TimeOnSite, and ChatUse.

Figure 9-12 shows separate table definitions for specialized classes. The relationship among Sale, InStoreSale, TelephoneSale, and OnlineSale is represented by the foreign key SaleID in all three specialized class tables. In each case, the foreign key representing the inheritance association also serves as the primary key of the table representing the specialized class. A similar situation exists for OnlineCart, ActiveCart, and OnReserveCart.

■ Enforcing Referential Integrity

In general terms, referential integrity is a constraint on database content; for example, “A sale must be to a customer” and “A sale item must be something that we stock in inventory.” For relational databases, the term **referential integrity** describes a consistent state between foreign key and primary key values. A referential integrity constraint requires that every foreign key value in one table must have a record in the associated table with that same value in the primary key.

The DBMS usually enforces referential integrity automatically once the schema designer identifies primary and foreign keys. For example, when a new row is added to a table containing a foreign key, the DBMS checks that the value

referential integrity every value as a foreign key in one table must have an equivalent value as the primary key in the associated table

FIGURE 9-12 Specialization classes of Sale and OnlineCart represented as separate tables

Table	Attributes
AccessoryPackage	AccessoryPackageID , AccessoryCategory, Description
AccessoryPackageContents	AccessoryPackageID , ProductItemID
CartItem	InventoryItemID , OnlineCartID , Quantity, CurrentPrice
Customer	AccountNumber , Name, MobilePhone, HomePhone, EmailAddress, Status
InventoryItem	InventoryItemID , ProductItemID , Size, Color, Options, QuantityOnHand, AverageCost, ReorderQuantity
OnlineCart	OnlineCartID , <i>CustomerAccountID</i> , StartDateTime, NumberOfItems, ValueOfItems, Status, ElapsedTime, HoldForDays
ActiveCart	OnlineCartID , ElapsedTime
OnReserveCart	OnlineCartID , HoldForDays
ProductComment	ProductCommentID , <i>ProductItemID</i> , <i>CustomerAccountNumber</i> , Date, Rating, Comment
ProductItem	ProductItemID , Gender, Description, Supplier, Manufacturer, Picture
PromoOffering	PromotionID , ProductItemID , RegularPrice, PromoPrice
Promotion	PromotionID , Season, Year, Description, StartDate, EndDate
Sale	SaleID , <i>CustomerAccountNumber</i> , SaleDateTime, PriorityCode, ShippingAndHandling, Tax, TotalAmount, MountainBucks
InStoreSale	SaleID , StoreID, RegisterID, ClerkID
OnlineSale	SaleID , TimeOnSite, ChatUse
TelephoneSale	SaleID , ClerkID, LengthOfCall
SaleItem	InventoryItemID , SaleID , Quantity, SoldPrice, ShipStatus, BackOrderStatus
SaleTransaction	SaleTransactionID , <i>SaleID</i> , Date, TransactionType, Amount, PaymentMethod

© Cengage Learning®

also exists as a primary key value in the related table and rejects the new row if no such primary key value exists. The database designer “tells” the DBMS which columns are foreign keys and which primary key columns they refer to by creating a **referential integrity constraint**. For example, a referential integrity constraint for SaleID in the SaleItem table would be written in SQL as:

```
ADD CONSTRAINT FK_SaleItem_Sale FOREIGN KEY
(SaleID) REFERENCES Sale(SaleID)
```

referential integrity constraint

a constraint, stored in the schema, that the DBMS uses to automatically enforce referential integrity

■ Database Normalization

Creating a set of database tables from a well-formed and accurate data model produces a correct and well-defined database. However, at times the data model may not have been built accurately or correctly. This could result in a database that is either incorrect, or not designed efficiently. In addition, sometimes developers will design a database without using a complete data model. In any event, it is a good practice to do a final review of the database, to ensure that it is well formed and correct. One important characteristic of a correct database design is that all of the tables in the database be normalized. A normalized database has certain characteristics that are important for accuracy and performance.

A normalized relational database schema has three important characteristics:

1. Allows flexibility in implementing future data model changes
2. Contains a minimum of redundant data
3. Prevents insertion, deletion, and update anomalies

A database schema is considered flexible and maintainable if changes to the database schema can be made with minimal disruption to existing data content and structure. For example, adding a new class to the schema shouldn't require redefining existing tables. Adding a new one-to-many association should only require adding new foreign keys and/or tables. Redundancy also plays a role in database longevity and usability. Excessive redundancy reduces schema flexibility and also reduces system performance. Anomalies in record updates, insertions, and deletions occur, for example, when deleting data in a table inadvertently deletes important but ancillary data. For example, if customer name and address information is maintained as part of a Sale, then deleting a Sale record might also completely delete the Customer information! Obviously, this is not a desirable result.

You have learned how to create a database schema based on a data modeling activity rigorous enough to create a correct and complete data model. You have also learned how to convert a conceptual data model to a relational model or relational database schema. The advantage of creating a database schema using this process is that the resulting schema is already in normalized form. However, in your career you will have occasion to work with databases that were not designed with this rigorous data modeling approach or in which data modeling errors were made, so you should understand how to evaluate and normalize a database.

Normalization is a formal technique for structuring a database to minimize data redundancy and to prevent data anomalies. It defines specific methods to eliminate redundancy and improve flexibility as well as to prevent data anomalies. Database theory defines increasing levels of normalization, including first normal form, second normal form, third normal form, fourth normal form, Boyce-Codd normal form, and others; however, a complete discussion of normalization is well beyond the scope of this textbook. The following sections briefly define the key concepts of first, second, and third normal forms. Usually ensuring that a database is in third normal form is sufficient for most business information systems.

■ First Normal Form

A table is in **first normal form (1NF)** if every field contains only one value. The formal way to state this is that all attribute values must be atomic. Non-atomic fields can be illustrated in two ways. Either you have attributes with multiple values in them, which are called multivalued attributes, or you have a table with multiple columns with the same name. **Figure 9-13(a)** illustrates the first approach, and **Figure 9-13(b)** illustrates the second. 1NF prohibits attributes such as *Dependent*, which is shown in Figure 9-13.

normalization a formal technique for transforming a relational schema to an equivalent one that minimizes data redundancy and eliminates data anomalies

first normal form (1NF) a restriction that all fields must be atomic, or single valued

FIGURE 9-13(a) Rows with multivalued attribute *Dependents*

SSN	Name	Department	Salary	Dependents
111-22-3333	Mary Smith	Accounting	40,000	John, Alice, Dave
222-33-4444	Jose Pena	Marketing	50,000	---
333-44-5555	Frank Collins	Production	35,000	Jan, Julia

FIGURE 9-13(b) Rows with varying number of *Dependent* columns

SSN	Name	Department	Salary	Dependent	Dependent	Dependent
111-22-3333	Mary Smith	Accounting	40,000	John	Alice	Dave
222-33-4444	Jose Pena	Marketing	50,000			
333-44-5555	Frank Collins	Production	35,000	Jan	Julia	

FIGURE 9-14 Schema normalized to first normal form

SSN	Name	Department	Salary
111-22-3333	Mary Smith	Accounting	40,000
222-33-4444	Jose Pena	Marketing	50,000
333-44-5555	Frank Collins	Production	35,000

RecordID	SSN	Dependent
1	111-22-3333	John
2	111-22-3333	Alice
3	111-22-3333	Dave
4	333-44-5555	Jan
5	333-44-5555	Julia

© Cengage Learning®

To correct this problem, the employee table must be divided into two tables. The first table will have the columns of SSN, Name, Department, and Salary. Another table, with a newly invented key, will have columns SSN and Dependent. Notice in the second table, that each row has a unique key, and also has a pairwise matching of SSNs and Dependents. Figure 9-14 illustrates the solution to the problem. The schema shown in Figure 9-14 is in first normal form.

Note that the correct data model for this situation would be an Employee class and a Dependent class with a one-to-many association. As stated previously, a correct data model—when converted to a set of database tables correctly—will result in a normalized schema.

■ Functional Dependency

functional dependency a relationship between columns such that the values in one (or more) column determine the values in a second column (or set of columns)

Before moving on to second and third normal forms, you should become familiar with the concept of a functional dependency. A **functional dependency** is a relationship between columns such that the values in one (or more) column determine the values in a second column (or set of columns). The association is formally stated as follows:

Attribute B is functionally dependent on attribute A if for each value of attribute A there is only one corresponding value of attribute B. Also written as FD: $A \rightarrow B$.

The most precise way to determine whether functional dependency exists is to pick two attributes in a table and insert their names as A and B of the previous statement and ask whether the result is true. For example, consider the attributes ProductItemID and Description in the ProductItem table (see Figure 9-15). ProductItemID is an internally invented primary key that is guaranteed to be unique within the table. To determine whether Description is functionally dependent on ProductItemID, substitute Description for attribute B and ProductItemID for attribute A in the functional dependency definition:

Description is functionally dependent on ProductItemID if for each value of ProductItemID there is only one corresponding value of Description.

Now ask whether the statement is true for all rows that could possibly exist in the ProductItem table. If the statement is true, then Description

FIGURE 9-15 RMO ProductItem table

ProductItemID	Gender	Description	Supplier	Manufacturer
10564	Both	Super Alpine Performance Skis	K2	K2
10766	Man	Extreme Ski Boots	Nordica	Nordica
1244	Man	Casual Chino Trousers	West Coast	Adida
1245	Man	Fleece Crew Sweatshirt	West Coast	Adida
1246	Man	Fleece Crew Sweatshirt V-Neck	West Coast	Adida
1247	Man	Fleece Crew Sweatshirt Zippered	West Coast	Adida
1248	Man	Solid Color Flannel Shirt	RMO	RMO
1249	Man	Plaid Flannel Shirt	RMO	RMO
1250	Man	Polo Shirt	RMO	RMO
1251	Man	Polo Shirt Zippered	RMO	RMO
1252	Man	Navigator Jacket	Colorado Supply	North Face
1253	Man	Navigator Jacket Hooded	Colorado Supply	North Face
1254	Man	Cotton Thermal Shirt	Colorado Supply	Under Armc

Source: Microsoft Corporation

is functionally dependent on ProductItemID. As long as the invented key ProductItemID is guaranteed to be unique within the ProductItem table, then the preceding statement is true. Therefore, Description is functionally dependent on ProductItemID. Note that the following notation is another way to write a functional dependency “FD: ProductItemID → Description.” Said another way, ProductItemID functionally determines Description.

Let’s check other columns. Look at ProductItemID and Supplier. Note that Supplier is also functionally dependent on ProductItemID, or FD: ProductItemID → Supplier. However, in this case, also notice that the reverse is not true. That Supplier does NOT functionally determine ProductItemID because given a value of Supplier, such as West Coast, you cannot determine a single value for ProductItemID.

Finally, note that ProductItemID → Gender, Description, Supplier, Manufacturer. In other words, because ProductItemID is the key field for this table, it functionally determines every other column in the rows. Thus, the ProductItem table is normalized correctly. The following sections look at second and third normal forms, including some situations where tables are not normalized completely and need to be corrected.

■ Second Normal Form

second normal form (2NF) a restriction that a table is in 1NF and that each non-key attribute is functionally dependent on the entire primary key

A table is in **second normal form (2NF)** if it is in 1NF and if each non-key attribute is functionally dependent on the entire primary key. A table violates 2NF when a non-key attribute is functionally dependent on only part of the primary key, which is only possible if the primary key contains multiple attributes. Thus, because the ProductItem table is in 1NF, it is also in 2NF because its primary key is a single column.

When a table’s primary key consists of two or more attributes, the analyst must examine functional dependency of non-key attributes on each portion of the primary key. For example, consider a modified version of the RMO PromoOffering table, as shown in **Figure 9-16**. Recall that this table represents a many-to-many association between Promotion and ProductItem. Thus, the table representing this association has a primary key consisting of the primary keys of Promotion (PromotionID) and ProductItem (ProductItemID).

To be in 2NF, each non-key attribute must be functionally dependent on the entire primary key consisting of the combination of PromotionID and ProductItemID. The simplest way to test for 2NF is to test for functional dependency of non-key attributes on each individual attribute of the primary key. Because the primary key contains two attributes, there are two statements that

FIGURE 9-16 Version of the RMO PromoOffering table that violates 2NF

PromotionID	ProductItemID	RegularPrice	PromoPrice
1	10564	\$599.99	\$529.99
1	10766	\$399.99	\$339.99
2	10564	\$599.99	\$449.00
2	10766	\$399.99	\$299.00
2	1250	\$49.99	\$29.00
2	1251	\$49.99	\$29.00

must be tested for each non-key attribute. To test RegularPrice attribute, these statements are:

RegularPrice is functionally dependent on PromotionID if for each value of PromotionID there is only one corresponding value of Regular Price.

RegularPrice is functionally dependent on ProductItemID if for each value of ProductItemID there is only one corresponding value of Regular Price.

If either statement is true—that is, an attribute is functionally dependent on only part of the key—then a 2NF violation exists. In this example, the first statement is false, but the second is true. RegularPrice depends on ProductItemID regardless of what promotions, if any, a product participates in. Another way to think about this example is to think about the underlying association represented by the PromoOffering table. A product can be part of multiple promotions at the same time. Although a product’s promotional price can be different in different promotions, its regular price is the same whether it participates in one promotion, three promotions, or none.

If a non-key attribute such as RegularPrice is functionally dependent on only part of the primary key, then you must remove the non-key attribute from its present table and place it in another table to satisfy the requirements of 2NF. Because the first functional dependency statement is true, RegularPrice belongs in a table that has ProductItemID as its primary key. If such a table doesn’t already exist, it must be created. However, in this case, there is already a table with ProductItemID as its primary key: the ProductItem table. Thus, the column RegularPrice is removed from the PromoOffering table and added to the ProductItem table, thus ensuring that the PromoOffering table is in 2NF.

Referring back to Figure 9-6, notice that the PromoOffering class was modeled incorrectly. It should be clear from our normalization exercise that the RegularPrice attribute really belongs in the ProductItem class and not the PromoOffering class. Normalization provides verification that the data model was constructed correctly.

■ Third Normal Form

third normal form (3NF) a restriction that a table is in 2NF and that no non-key attribute is functionally dependent on any other non-key attribute

A table is in **third normal form (3NF)** if it is in 2NF and if no non-key attribute is functionally dependent on any other non-key attribute. To verify that a table is in 3NF, you must check the functional dependency of each non-key attribute against every other non-key attribute. This can be cumbersome for a large table because the number of pairs that must be checked grows quickly as the number of non-key attributes grows. The number of functional dependencies to be checked is $N \times (N - 1)$, where N is the number of non-key attributes. Note, too, that functional dependency must be checked in both directions (i.e., A dependent on B, and B dependent on A).

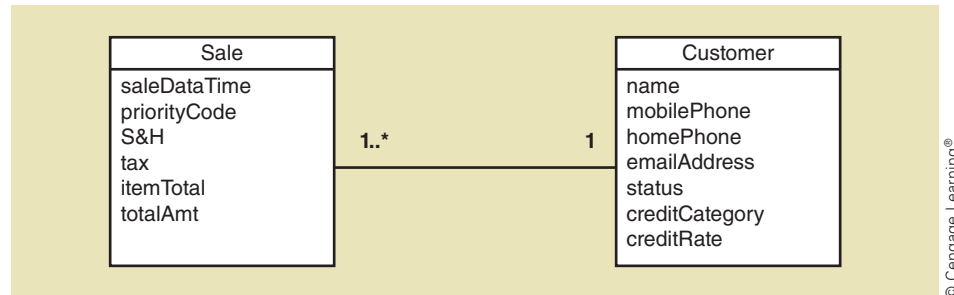
FIGURE 9-17 Sale class with *itemTotal* and *totalAmt*

FIGURE 9-18 Sale table modified to violate 3NF

SaleID	SaleDate1	PriorityC	Shipping	Tax	ItemTotal	TotalAmt	CustomerAccountN
841152	9/1/2012		\$8.50	\$0.00	\$91.35	\$99.85	134425
841153	9/2/2012		\$6.00	\$0.00	\$28.00	\$34.00	187763
*			\$0.00	\$0.00	\$0.00		

Record: 1 of 2 | No Filter | Search

Source: Microsoft Corporation

This section demonstrates two common violations of 3NF. To illustrate these violations, the Sale class and the Customer class in the class diagram have been modified. **Figure 9-17** illustrates these modified classes.

A common example of a 3NF violation is an attribute that can be computed by a formula or algorithm using other table values as inputs. Common examples of computable attributes include subtotals, totals, and taxes. To illustrate this concept, the Sale class was modified to include an *itemTotal* attribute.

The modified Sale table is shown in **Figure 9-18**. The *ItemTotal* field is the sum of *SoldPrice* amounts from the *SaleItem* table. The *TotalAmt* field is the sum of the *Shipping*, *Tax*, and *ItemTotal*, as follows:

$$\text{TotalAmt} = \text{Shipping} + \text{Tax} + \text{ItemTotal}, \text{ or}$$

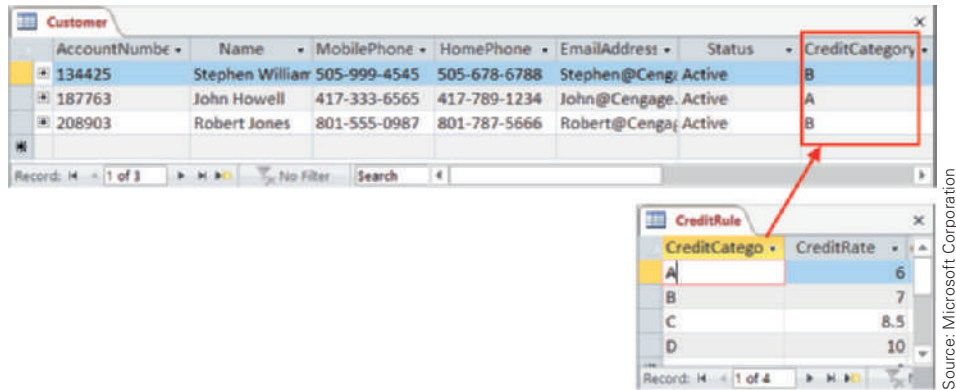
FD: Shipping, Tax, ItemTotal → TotalAmt

Computational dependencies are a form of redundancy because a change to the value of any input variable in the computation (e.g., *Shipping*) also changes the result of the computation (i.e., *TotalAmount*). The way to correct this type of 3NF violation is simple: Remove the computed attribute from the database. Eliminating the computed attribute from the database doesn't mean that its value is lost. It can always be recalculated when it is needed.

The other example in the Customer class is the addition of a *creditCategory* field along with its associated *creditRate* (refer back to **Figure 9-17**). In other words, a *creditCategory* of A has a *creditRate* of 6 percent, and a *creditCategory* of B has a *creditRate* of 7 percent. It is tempting to put all this data right in the Customer table, so that you can see the credit rating of a customer and the interest rate that she must pay. However, it should be obvious that this is a bad design. If the interest rate for *CreditCategory* B changed to 8 percent, there may be hundreds or thousands of places where that value is stored in the Customer table, then requiring an update. The solution to this problem is to make a new table, called *CreditRule*, with *CreditCategory* and *CreditRate* as the fields and *CreditCategory* as the key. The Customer table will have a foreign key that points to the *CreditCategory* field in the *CreditRule* table. **Figure 9-19** shows the solution with the two tables defined.

From time to time, a database design will relax 3NF requirements and have some tables that are not in 3NF compliance. A common example is a customer address. In the address field, it is normal to have both state and zipcode fields.

FIGURE 9-19 Customer table in 3NF showing CreditRule table



Source: Microsoft Corporation

However, because zip codes do not cross state lines, the state field is functionally determined by the zipcode field. However, because the zipcode values almost never change, to avoid having a zipcode table, and due to convention, most databases permit both state fields and zipcode fields in the address information of a single table.

■ Data Types

data type the storage format and allowable content of a program variable, class attribute, or relational database field

primitive data types data types supported directly by computer hardware or a programming language

complex data types combinations of or extensions to primitive data types that are supported by programming languages, operating systems, and DBMSs

A **data type** defines the storage format and allowable content of a program variable, class attribute, or relational database attribute or column. **Primitive data types** are supported directly by computer hardware and programming languages and include integers, single characters, and real numbers (floating-point numbers). **Complex data types** are combinations of or extensions to primitive data types that are supported by programming languages, operating systems, and DBMSs. Examples include arrays and tables, strings (character arrays), dates, times, currency (money), audio streams, still images, motion video streams, and Uniform Resource Locators (URLs or Web links).

A database designer must choose an appropriate data type for each attribute in a relational database schema. For many attributes, the choice of a data type is relatively straightforward. For example, designers can represent customer names and addresses as strings, inventory quantities as integers, and item prices as currency. RDBMSs support a variety of primitive and complex data types required by modern information systems. **Figure 9-20** contains a partial listing of some of the data types available in the Microsoft SQL Server RDBMS. The *varbinary* data type is typically used to store such data items as pictures, sound, and video encoded in such standardized formats as JPEG, MP3, and MP4.

FIGURE 9-20 Examples of data types available in the Microsoft SQL Server RDBMS

Type(s)	Description
datetimeoffset	Date, time, and time zone
int, small int, and bigint	Whole numeric values
float and real	Numeric values with fractional quantities
money	Currency values and related symbols (e.g., \$ and €)
nchar and nvarchar	Fixed- and variable-length Unicode string
varbinary	Variable-length byte sequence up to 2GB
xml	XML document up to 2GB

© Cengage Learning®

■ Distributed Database Architectures

Earlier in the chapter, you learned about various database configurations, from local desktop to LAN server to a large database server farm, and finally to distributed server farms residing in multiple data centers. As an organization becomes very large, with a user base that spans the globe, it becomes more beneficial to locate the data at locations that are closer to the users. There are three scenarios for distributing the data:

decentralized database a database stored at multiple locations without needing to be interconnected through a network or synchronized

homogeneous distributed database a database distributed across multiple locations with the same DBMS, and all database access coordinated by a global schema

heterogeneous distributed database a database distributed across multiple locations with different DBMSs and with local access allowed without global schema coordination

- **Decentralized database.** In some situations, a global company may have pockets of database users who share data with each other, but do not have the need to share the data globally. Each database may have the same structure and schema as the other databases, but does not need to be connected with Internet or wide area networks (WANs). In this situation, the data is purely local, even though the database configuration may be the same.
- **Homogeneous distributed database.** In the situation where the data need to be shared or at least available throughout the reach of the organization, a homogeneous distributed database is the correct structure. **Figure 9-21** illustrates a typical configuration for this type of environment. The global schema contains the metadata about which data reside at which node and directs queries and updates appropriately. In this configuration, the global schema is available to every database user issuing database queries. Normally, a homogeneous configuration will use the same DBMS at each location.
- **Heterogeneous distributed database.** This configuration combines the features of the two previous configurations, namely, that there are some users and queries that are purely local, combined with other users and queries that require global access. With a heterogeneous configuration, different locations may also utilize distinct DBMSs, especially where most of the queries are local and only occasional queries require global interfaces. **Figure 9-22** illustrates a typical heterogeneous distributed database configuration.

FIGURE 9-21 Typical homogeneous distributed database configuration

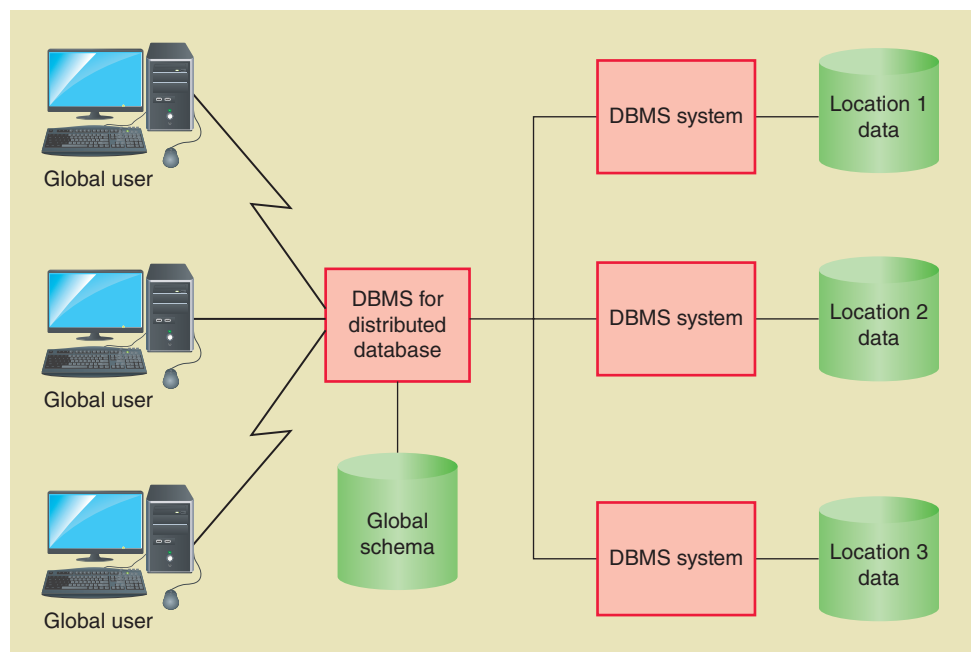
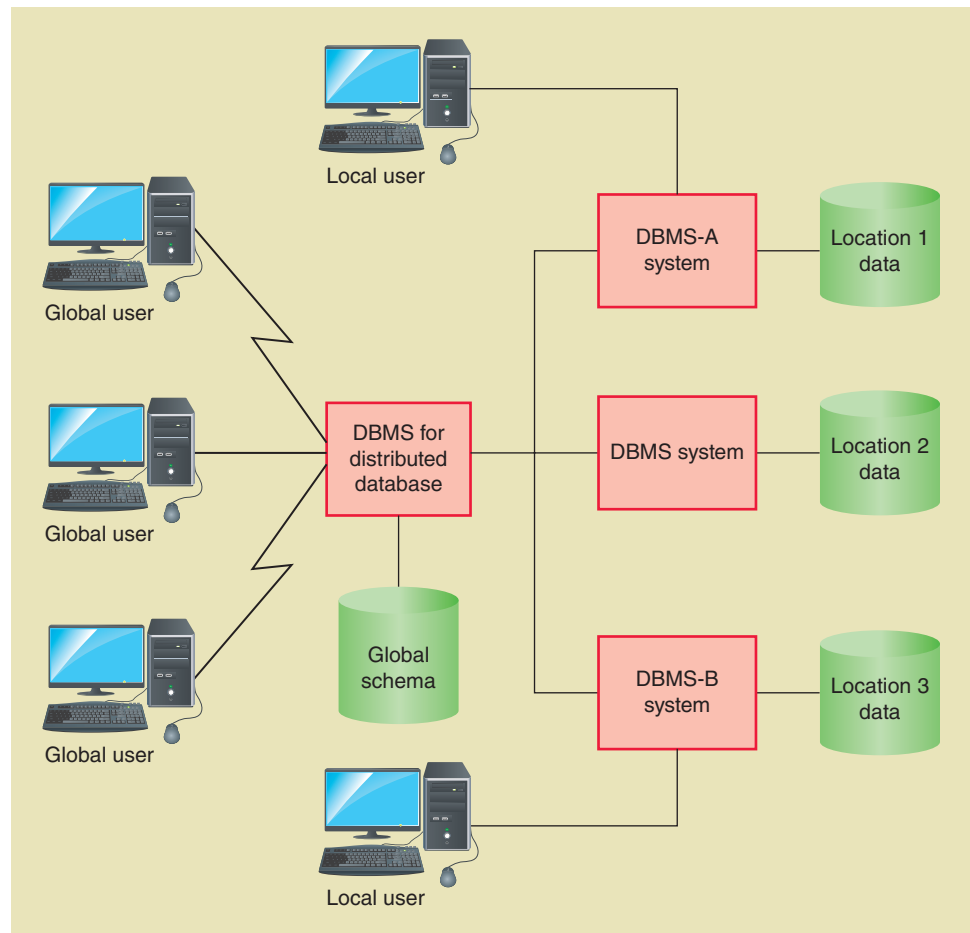


FIGURE 9-22 Typical heterogeneous distributed database configuration



■ Implementation Approaches for Distributed Database

Once the need for a distributed database is recognized, decisions must be made on how to partition the database and what the configuration will be at each location. Obviously, this is a complex decision that requires careful analysis of the data requirements for each location. Configuration decisions must consider such things as frequency of queries versus updates; size and frequency of result sets; data transmission costs; physical location of server farms; and many other cost, efficiency, and performance requirements. The following sections briefly consider four distribution strategies:

1. Data replication
2. Horizontal partitioning
3. Vertical partitioning
4. Combinations of the above three strategies

■ Data Replication

Complete database copies are hosted at each location or server farm and maintained by cooperating DBMSs. The servers are usually distributed across geographic locations. Application programs can access any server and usually make database updates to only one server. Servers at distinct locations periodically exchange update information to synchronize their database copies.

Applications can direct access requests to any available server, with preference given to the nearest server. When a server is unavailable, clients can redirect requests to another available server. In spite of these advantages, replicated

database synchronization updating one database copy with changes made to other database copies

database servers do have some drawbacks. When data are updated on one database copy, clients accessing that same data from another database copy receive an outdated response. To counteract this problem, each database copy must periodically be updated with changes from other database servers. This process is called **database synchronization**.

The time delay between an update to a database copy and the propagation of that update to other database copies is an important database design decision. During the time between the original update and the update of database copies, application programs that access outdated copies aren't receiving responses that reflect current reality. Designers can address this problem by synchronizing more frequently or continuously. However, synchronization then consumes a substantial amount of database server capacity, and a large amount of network capacity among the related database servers must be provided. The proper synchronization strategy is a complex trade-off among cost, hardware and network capacity, and the need of application programs and users for current data.

The advantage of replicated distributed databases is that each location is essentially independent for any given query or update; thus, response times tend to be fast. Also, in case of unavailability for any location, the other locations are immediately prepared to handle queries. In addition, transaction processing, which requires multiple updates to occur, is straightforward because only one location is impacted. The two major disadvantages are the increased storage requirements because the database is duplicated multiple times, and, as mentioned previously, keeping all the locations synchronized.

■ Horizontal Partitioning

Horizontal partitioning of the database occurs when a table is split by storing some rows or records at one location and other rows or records at another location. For example, suppose a large international bank had branches around the world with one data center in the United States, a second data center in Canada, a third data center in Europe, and so forth. It would make sense to partition the CustomerAccount tables, summary, and detail information horizontally so that the data for U.S. customers resided in the U.S. data center. **Figure 9-23** illustrates a global view one of the CustomerAccount tables, showing how it might be partitioned. The global schema will identify which records reside at which location so queries and updates can be routed to the correct data center. For example, a person traveling internationally may need to check her information from various locations around the world. Even though this is a fairly straightforward method to partition the database, it does get more complex as customers also become global and have accounts or loans in multiple locations throughout the world. Hence, even with horizontal partitioning there may be the need to maintain duplicate data or to synchronize data.

To reconstruct the complete base tables requires that data from all locations be combined together. This situation might be required for combined reporting or data analysis processing.

FIGURE 9-23 Horizontally partitioned table of a database

AcctNumb	LastName	FirstName	SSN	TypeOfAcct	Balance	DateLastActivity	
01-85562-1	Jones	Bill	878-77-9890	Checking	\$ 7,908.39	5/9/2014	U.S. accounts
01-85444-2	Johnson	Harold	676-44-3433	Checking	\$25,698.33	5/2/2013	
02-45443-2	Williams	Jonathon	343-44-2322	Checking	\$ 3,938.77	4/4/2012	
01-34999-1	Redd	Mary	898-79-3487	Savings	\$12,898.71	12/2/2013	
01-23989-2	Chun	Tun	233-59-6765	Savings	\$ 8,932.67	1/8/2014	Hong Kong accounts
01-87889-4	Gang	Bao	322-48-3545	Checking	\$ 568.33	3/4/2014	
01-32339-2	Jiang	Rui	550-43-5454	Savings	\$35,788.23	7/8/2014	
02-39988-1	Ma	Shuo	343-98-2345	Checking	\$ 1,893.55	8/23/2014	

Source: Microsoft Corporation

The major advantage of this configuration is its simplicity. Each data center is essentially independent and database access can be optimized as though it were a local data center. Security is also handled at the local level. A disadvantage is that, because there is no duplication, this configuration is subject to unavailability should one of the locations experience a catastrophe.

■ Vertical Partitioning

Vertical partitioning of the database occurs when complete tables or only specific columns of a base table are stored at distinct locations. A vertically partitioned table may have some columns at location A, other columns at location B, and other columns at location C. Distributing entire base tables to distinct locations is fairly straightforward. Distributing only columns of the same table to distinct locations is more complex, and is more complex than horizontal partitioning. So when would an organization implement vertical partitioning?

Figure 9-24 illustrates vertical partitions for an AssemblyPart table in a Supply Chain database for a company that obtains parts from manufacturers in one location, such as Japan, and does final inspection and assembly of those parts in another location, such as the United States. For the AssemblyPart table in Japan, information is required about the description, the manufacturer, and the quantity on hand of these raw parts. For the AssemblyPart table in the United States, information is required about the description, the schematic number of the assembly instructions, the document number of the inspection instructions, and the quantity on hand at the assembly plant. Thus, vertical partitioning of the database occurs more frequently when different functional areas must access the same database and the same data elements.

■ Combination of Replication, Horizontal, and Vertical Partitions

All of the preceding three techniques can be combined to provide the right data at the right place in the right form. A large international organization will often have different functional areas located at distinct geographical locations, requiring vertical partitioning. Other functional areas, such as Sales, may require customer and sales information at all locations, which may best be partitioned in a horizontal configuration. Some information, such as corporate-wide information, may need to be available at all locations and may utilize replicated tables. The design and maintenance of this complex configuration would be designed under the direction of the data administrator and the database administrator roles, as explained previously. A single project team would not take responsibility for designing and administering this scenario.

FIGURE 9-24 Vertical partition of the AssemblyPart table

PartNumber	Description	Manufacturer	QtyOnHand	SchematicNo	InspectionNo	QtyOnHand2
4568-AC9	Screw assembly	Westco Inc	348	42-596	56	346
7618-IF44	Handle assembly	Japan Tools	276	16-443	43	434
7678-AD22	Door1 assembly	Tokyo Hardware	58	76-454	65	765
4890-XX88	Door2 assembly	Tokyo Hardware	97	78-443	34	446
9890-CD87	Interior module	Open Electronics	454	23-794	67	454
6766-DY65	Interior seal assembly	Sealants Inc	611	56-545	23	2132
8769-DD77	Connection assembly	Open Electronics	546	90-787	22	722
2311-AB28	Crank assembly	Westco Inc	768	33-571	12	121
3432-R888	Double pulley assembly	Westco Inc	564	90-443	43	342

Source: Microsoft Corporation

■ RMO Distributed Database Architecture

The starting point for designing a distributed database architecture is information about the data needs of geographically dispersed users. Some of this information for RMO was gathered as an analysis activity and is summarized here:

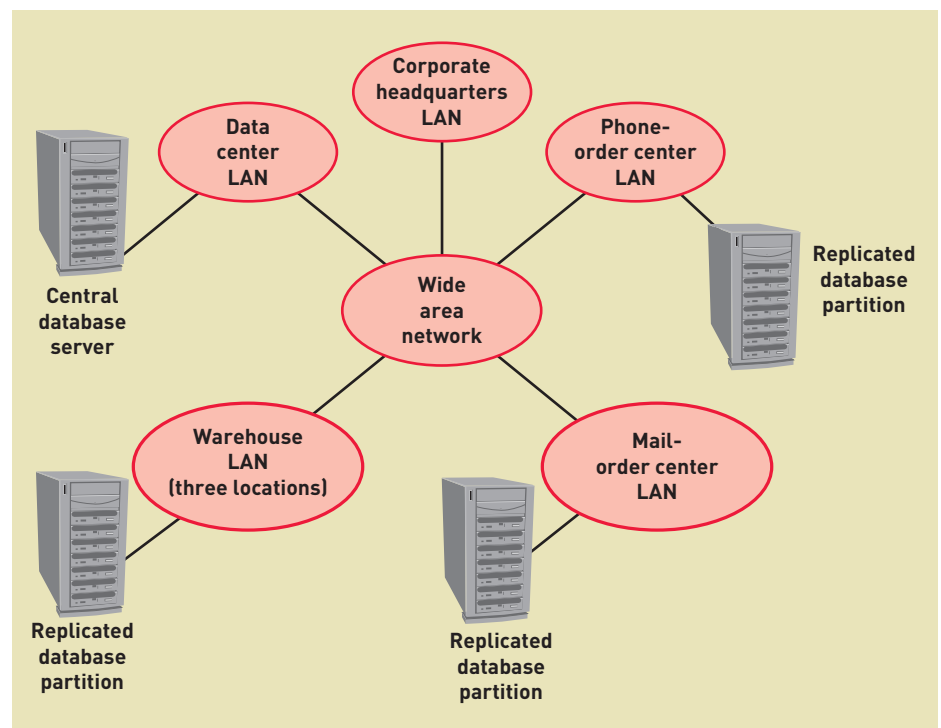
- Warehouse staff members (Portland, Salt Lake City, and Albuquerque) need to check inventory levels, query orders, record back orders and order fulfillment, and record order returns.
- Phone-order staff members (Salt Lake City) need to check inventory levels; create, query, update, and delete orders; query customer account information; and query catalogs.
- Customers using the online sales system and sales associates in retail stores need the same access capabilities as phone-order staff.
- Marketing staff members (Park City) need to query and adjust orders, query and adjust customer accounts, and create and query product and promotion information.

RMO has already decided to manage its database by using the existing server cluster in the Park City data center. That same center will also host servers supporting the online sales, order fulfillment, and marketing systems. Thus, a high-capacity wide area network (WAN) will be needed to connect the servers to local area networks (LANs) in the warehouses, phone-order centers, retail stores, headquarters, and data centers.

An architecture that stores the entire database on a single server is not feasible for the Consolidated Sales and Marketing System. There are many accesses from many locations at many different times. Inevitable database server downtime would result in lost productivity, sales, and reputation. In essence, the entire company would grind to a halt, and its future revenue stream would be jeopardized. As for many modern organizations, that risk is simply too much to bear.

A more complex alternative that addresses the risk is shown in **Figure 9-25**. Each remote location employs a combination of database partitioning and replication. A server at each warehouse stores a local copy of the order and inventory

FIGURE 9-25 *Replicated and partitioned database server architecture for RMO*



portions of the database. Servers in the phone-order center and retail stores host local copies of a larger subset of the database. Corporate headquarters relies on the central database server in the data center.

The primary advantages of this architecture are fault tolerance and reduced WAN capacity requirements. Each location could continue to operate independently if the central database server failed. However, as the remote locations continued to operate, their database contents would gradually drift out of synchronization.

The primary disadvantages to the distributed architecture are cost and complexity. The architecture saves WAN costs through reduced capacity requirements, but adds costs for additional database servers. The cost of acquiring, operating, and maintaining the additional servers would probably be much higher than the cost of adding greater WAN capacity.

So, does the proposed architecture make sense for RMO? The answer depends on some data that hasn't yet been gathered and on answers to some questions about desired system performance, tolerances for downtime, cost and reliability of WAN connections, and the cost and availability of cloud services. RMO management must also determine its goals for system performance and reliability. The distributed architecture would provide higher performance and reliability but at substantially increased cost. Management must determine whether the extra cost is worth the expected benefits.

■ Protecting the Database

In Chapter 6, you learned about the need to consider security issues early in the development of the new system. In the discussion of access controls in Chapter 6, you learned about the process of authentication and authorization. Those concepts apply directly to the protection of the data in the database and are used extensively before allowing access to the database. Additional techniques, such as data encryption, are also a frequently used technique to keep the data protected from unauthorized persons and programs.

The responsibility for designing and deploying a secure database is shared between the project team, that is, the system development team, and the DBA. The DBA provides advice and direction to the project team as they design the new system to ensure that proper controls and mechanisms are included in the code of the new system. The DBA then has responsibility for protecting the DBMS and the database when it is put into production and while it is operational. In Figure 9-1, you see that there are three types of agents that have access to the database: the application programs, database users, and database administrators. The development team plays a major role in the security aspects of the application programs. The DBA is concerned about security for all three types of agents.

In today's computing environment where databases are available to the entire world via the Internet, you will frequently read about occasions when a particular database has been compromised and private data has been exposed. Protecting the database is a serious issue in the development and deployment of information systems, and a thorough treatment of the subject is well beyond the scope of this textbook. There are many books and even university courses that cover security for information systems, networks, and databases.

In addition to the security issues for protecting the database, there is the need to protect the data in the database from catastrophes ranging from simple power outages to major natural disasters. The following sections provide an introduction to some of the concepts, including programming issues, related to transaction logging and complex updates. In Chapter 6, you were also introduced to the need for backup and recovery.

transaction logging a technique by which any update to the database is logged with such audit information as user ID, date, time, input data, and type of update

■ Transaction Logging

Transaction logging is a technique by which any update to the database is logged with such audit information as user ID, date, time, input data, and type of update. Transaction logging provides a record of database changes that is stored in a separate location and can be checked independently of the database itself. The fundamental idea is to create an audit trail of all database updates and, therefore, track any errors or problems that occur. Most DBMSs include transaction logging as part of the DBMS software, although database designers or administrators can customize its application.

Transaction logging achieves two objectives. First, it helps discourage fraudulent transactions or malicious database changes. For example, if a person knows that his or her ID will be associated with every check request, that person isn't likely to request a bogus payment. Similarly, a disgruntled employee who might be tempted to delete important records knows that his or her actions are being logged.

Second, a logging system provides a recovery mechanism for erroneous transactions. A midlevel logging system maintains the set of all updates. The system can then recover from errors by “unapplying” the erroneous transactions. More sophisticated logging systems can provide “before” and “after” images of the attributes or rows that are changed by the transaction as well as the audit trail of all transactions. These sophisticated systems are typically used only for highly sensitive or critical data files, but they do represent an important control mechanism that is available when necessary.

■ Concurrency and Complex Update Controls

Databases provide the foundation of almost every activity that is performed using an electronic device. Activities such as making a call on a cell phone, shopping online, sending a text, playing an Internet game, sharing your photos online, accessing your Facebook account, checking the price of your stocks and bonds all depend on databases and database technology. As you can imagine, large, active databases often have tens of thousands of users every minute who need to read and update the data. Not only do databases need to be extremely fast, but they also need to be designed so that multiple users do not damage each other's data inadvertently when they are accessing or updating the same information.

In some situations, such as looking at your online photos, users will be accessing personal or private information that only they will be using. However, in many situations, such as online shopping, the same information can be desired by thousands of users at the same time—some of which will only look at it, but others will need to update it. For example, when you purchase something on Amazon, the inventory levels and availability information must be updated immediately. It is not uncommon for multiple customers to be viewing and updating the information at the same time.

One of the problems that potentially can occur when multiple people are updating the database at the same time is called the “lost update problem.” For example, suppose Bill wants to buy some shoes. He accesses the record that contains shoe inventory levels. It indicates that there are 10 pairs in stock. Gary also wants to buy the same shoes, and also reads the shoe information. Bill buys a pair of shoes, and the system decrements the quantity and rewrites the quantity to nine pairs left. Gary does the same, and the system also rewrites the quantity to nine pairs. Unfortunately, Bill's update was overwritten by Gary's and so it becomes a “lost update.”

To solve this problem, techniques that implement two concepts are used. This section introduces the concepts, but does not explain all the techniques. The details of the techniques are beyond the scope of this text and fit more comfortably in a database textbook.

transaction a piece of work with several steps that must all be completed to be valid

The first concept is of a **transaction**. A transaction is a piece of work that has several steps, including several reads and writes to the database, that must all be completed to be valid. An example might be buying movie tickets online. The transaction will require the following steps:

1. Read the database for available seats and display it.
2. Update the database when the user chooses seats.
3. Start a timer.
4. Accept user purchase information (credit card number).
5. Verify credit and send a charge to the user's credit card.
6. Post the user's payment to the company ledger.
7. Permanently update the seat database that the chosen seats are now taken.
8. Update the user information with a reservation for the specific seats.

All of these steps must be completed for the “purchase ticket transaction” to be complete. If the user stops, if the credit card charge is refused, or if the timer runs out, then all the prior changes must be undone, and the database returned to its original condition. This is a rather long transaction involving a considerable period of time and many steps.

database lock the technique of applying exclusive control to a part of the database so that one user at a time can use the data

The second important concept that is widely used to protect against lost updates is to apply a **database lock**. A database lock is the technique where a portion of the database is locked by one user so that no other users (or programs) can use that portion of the database until the first user is finished and releases the lock.

The portion of the database that is locked can vary and is usually a design issue decided by the DBA. Locks can be applied to individual columns, to single records or table rows, to an entire table, or even to the entire database. Locking small portions of the database, such as a column or even a row, requires more sophistication to track it and control it. However, it is more efficient because other users can still use most of the database. Locking large portions, such as an entire table or even the database, is easy to implement, but makes the database inefficient for a multiple-user environment. The DBA decides the best type of locking to use depending on the use and activity level of the database.

shared or read lock a lock where other transactions are allowed to read the data

exclusive or write lock a lock where no other activity is allowed, neither reading nor writing the data

There are two types of locks that are used, a **shared lock**, also called a **read lock**, and an **exclusive lock**, also called a **write lock**. If the user application only needs to read the data, with no need to update it, it can issue a shared lock. A shared lock allows other users to also read the same data, but not to update it. If the user application needs to read and then rewrite the data, then an exclusive lock is issued. No other user can even read the data that has an exclusive lock.

CHAPTER SUMMARY

Most modern information systems store data in a database and access and manage the data by using a DBMS. The actual design of the database schema is a responsibility of the project team, with input from other technical staff, such as the DA and the DBA. One of the key activities of systems design is developing a relational database schema. A relational database is a collection of data stored in tables. A relational database schema is normally developed from a domain class diagram. Each class is represented as a separate table. One-to-many associations are represented by embedding foreign keys in class tables. Many-to-many

associations are represented by creating additional tables containing foreign keys of the related classes.

As organizations grow and become global, there is frequently a need to distribute the database across multiple data centers in distinct physical locations. Methods of partitioning the database, either horizontally or vertically, are used to provide an effective, yet reliable distribution of data between the data centers.

Because stored data is such an important organizational asset, database design incorporates security measures to ensure the correctness, completeness, and security of data.

KEY TERMS

attribute	exclusive lock or write lock	referential integrity constraint
attribute value	first normal form (1NF)	relational database management system (RDBMS)
candidate key	foreign key	row
complex data types	functional dependency	schema
data administrator (DA)	heterogeneous distributed database	second normal form (2NF)
data type	homogeneous distributed database	shared lock or read lock
database (DB)	key	Structured Query Language (SQL)
database administrator (DBA)	normalization	tables
database lock	primary key	third normal form (3NF)
database management system (DBMS)	primitive data types	transaction
database synchronization	referential integrity	transaction logging
decentralized database		

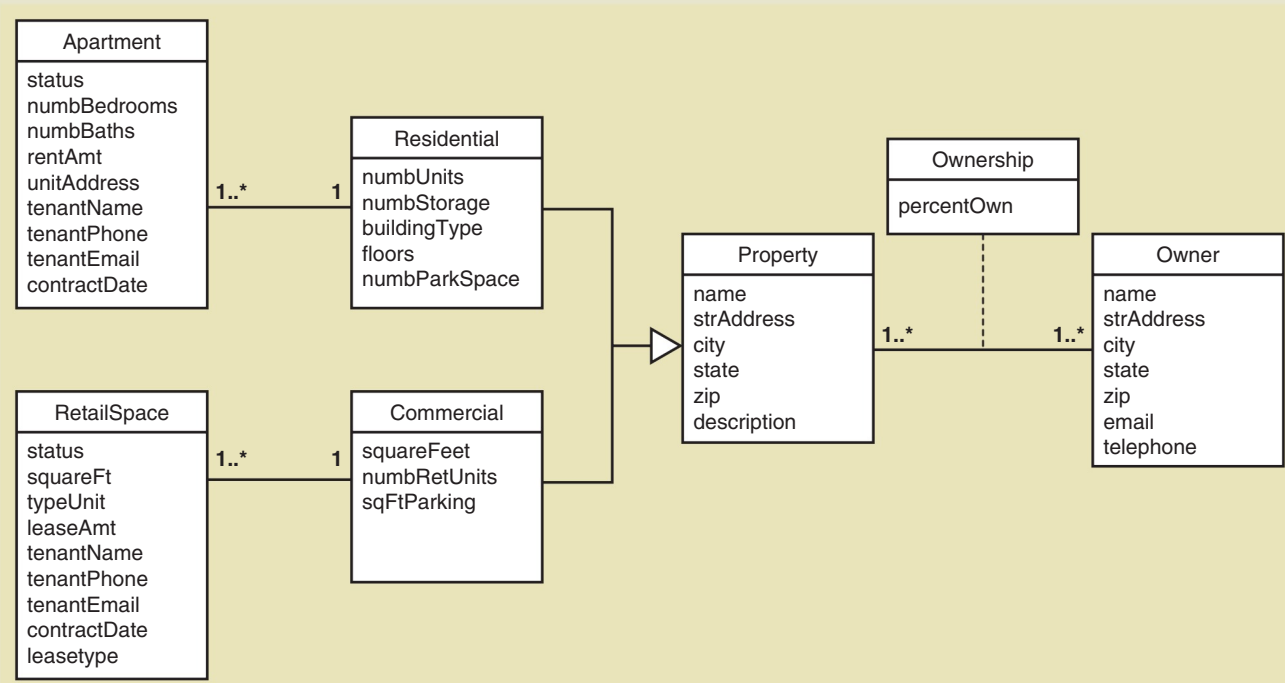
REVIEW QUESTIONS

- List the components of a DBMS and describe the function of each.
- What is a database schema? What information does it contain?
- What does SQL stand for and what is its purpose?
- Why are databases the preferred method of storing data used by an information system?
- What are the responsibilities of a data administrator?
- What are the responsibilities of a database administrator?
- With respect to relational databases, briefly define the terms *row* and *attribute value*.
- What is a primary key? Are duplicate primary key values allowed? Why or why not?
- What is the difference between a natural key and an invented key? Which type is most commonly used in business information processing?
- What is a foreign key? Why are foreign keys used or required in a relational database? Are duplicate foreign key values allowed? Why or why not?
- Describe the steps used to transform a domain class diagram into a relational database schema.
- What is referential integrity? Describe how it is enforced when a new foreign key value is created, when a row containing a primary key is deleted, and when a primary key value is changed.
- What types of data (or attributes) should never be stored more than once in a relational database? What types of data (or attributes) usually must be stored more than once in a relational database?
- What is relational database normalization? Why is a database schema in third normal form considered to be of higher quality than a non-normalized database schema?
- Describe the process of relational database normalization. Which normal forms rely on the definition of functional dependency?
- What is the difference between a primitive data type and a complex data type?
- What additional database management complexities are introduced when database contents are replicated in multiple locations?
- When should database design be performed? Can the database be designed iteratively or must the entire database be designed at once?
- What is the basic purpose of transaction logging?
- What is the difference between homogeneous distributed database and a heterogeneous distributed database?
- What is the purpose of database synchronization for a replicated database?
- What are the advantages and disadvantages of replicated databases?
- Which would be easier to configure and maintain? A vertically partitioned database or a horizontally partitioned database? Explain why.
- What is a transaction with regard to updating a database?
- What is transaction logging and what is its purpose?
- What is the difference between a shared lock and an exclusive lock?
- What is another name for an exclusive lock?

PROBLEMS AND EXERCISES

1. The Universal Product Code (UPC) is a bar-coded number that uniquely identifies many products sold in the United States. For example, all printed copies of this textbook sold in the United States have the same UPC bar code on the back cover. Now consider how the design of the RMO database might change if each individual item sold by RMO were required by law to carry a permanently attached UPC (e.g., on a label sewn into a garment or on a Radio Frequency Identification [RFID] tag attached to an item). How might the RMO relational database schema change under this requirement?
2. Assume that RMO will begin asking a random sample of customers who order by telephone about purchases made from competitors. RMO will give customers a 15 percent discount on their current order in exchange for answering a few questions. To store and use this information, RMO will add two new classes and three new associations to the class diagram. The new classes are *Competitor* and *ProductCategory*. *Competitor* has a one-to-many association with *ProductCategory*, and the existing *Customer* class also has a one-to-many association with *ProductCategory*. *Competitor* has a single attribute called *Name*. *ProductCategory* has four attributes: *Description*, *DollarAmountPurchased*, *MonthPurchased*, and *YearPurchased*. Revise the relational database schema shown in Figure 9-10 to include the new classes and associations. All tables must be in 3NF.
3. Assume that RMO will use a relational database, as shown in Figure 9-11. Assume further that a new catalog group located in Milan, Italy, will now create and maintain the product catalog. To minimize networking costs, the catalog group will have a dedicated database server attached to its LAN. Develop a plan to partition the RMO database. Which tables should be replicated on the catalog group's local database server? Update Figure 9-25 to show the new distributed database architecture.
4. Visit the Web site of an online catalog vendor similar to RMO (such as www.llbean.com or www.landsend.com) or an online vendor of computers and related merchandise (such as www.cdw.com or www.newegg.com). Browse the online catalog and note the various types of information contained there. Construct a list of complex data types that would be needed to store all the online catalog information.
5. **Figure 9-26** illustrates a partial class diagram for a property management company. Using the data in the class diagram, create a database schema.

FIGURE 9-26 Partial class diagram for a property management company



6. Given the database table in **Figure 9-27** of university course and sections offered, normalize the table so that it is in third normal form. Hint: Look for functional dependencies.
7. Given the database table in **Figure 9-28** of employees and their employment, normalize the table so that it is in third normal form. Hint: Look for functional dependencies.

FIGURE 9-27 Non-normalized Course-section table

Course	Section	Course Name	Room#	#Chairs	#Enrolled	Time	Teacher	Students
IS205	001	Intro to IS	TB105	55	46	3:00 MWF	B. Jones	R. Smith H. Black P. Harris A. Wells
IS401	001	Systems Analysis & Design	TB223	45	30	1:00 MWF	R. White	M. Tams N. Rich T. Nells

© Cengage Learning®

FIGURE 9-28 Non-normalized Employee table

Emp#	Employee Name	Job Title	Wage Range	Date Promoted	Supervisor Emp#	Supervisor Name
876	W. Johnson	Machinist Pipe Fitter Worker	25.00–45.00 18.00–30.00 12.00–25.00	June 1, 2011 May 15, 2006 July 1, 2002	450	B. Noch
651	A. Hansen	Welder Worker	20.00–35.00 12.00–25.00	Aug 1, 2012 June 15, 2009	335	P. Williams

© Cengage Learning®

CASE STUDY

Computer Publishing Inc.

In only a decade, Computer Publishing Inc. (CPI) grew from a small textbook publishing house into a large, international company with significant market share in traditional textbooks, electronic books, and distance education courseware. CPI's processes for developing books and courseware were similar to those used by most other publishers, but those processes had proven cumbersome and slow in an era of rapid product cycles and multiple product formats.

Text and art were developed in a wide variety of electronic formats, and conversions among those formats were difficult and error-prone. Many editing steps were performed with traditional paper-and-pencil methods. Consistency errors within books and among books and related products were common. Developing or revising a book and all its related products typically took a year or more.

CPI's president initiated a strategic project to reengineer the way that CPI developed books and related products. CPI formed a strategic partnership with Davis Systems (DS) to develop software that would support the reengineered processes. DS had significant experience developing software to support product development in the chemical and pharmaceutical industries by using the latest development tools and techniques, including object-oriented software and relational databases. CPI expected the new processes and software to reduce development time and cost. Both companies expected to license the software to other publishers within a few years.

A joint team specified the workflows and high-level requirements for the software. The team developed plans for a large database that would hold all book and courseware content through all stages of production. Authors, editors, and other production staff would interact with the database in a variety of ways, including traditional word

processing programs and Web-based interfaces. When required, format conversions would be handled seamlessly and without error. All content creation and modification would be electronic—no text or art would ever be created or edited on paper, except as a printed book ready for sale.

Software would track and manage content through every stage of production. Content common to multiple products would be stored in the database only once. Dependencies within and across products would be tracked in the database. Software would ensure that any content addition or change would be reflected in all dependent content and products, regardless of the final product form. For example, a sentence in Chapter 2 that refers to a figure in Chapter 1 would be updated automatically if the figure were renumbered. If a new figure were added to a book, it would be added automatically to the related courseware presentation slides. Related courseware and study material on the Web site would automatically

reflect changes, such as a new answer to an end-of-chapter question.

1. Consider the contents of this textbook as a template for CPI's database content. Draw a class diagram that represents the book and its key content elements. Expand your diagram to include related product content, such as a set of PowerPoint slides, an electronic book formatted as a Web site or PDF file, and a Web-based test bank.
2. Develop a list of data types required to store the content of the book, slides, and Web sites. Are the relational DBMS data types listed in Figure 9-20 sufficient?
3. Authors and editors are often independent contractors, not publishing company employees. Consider the implications of this fact for controls and security. How would you enable authors and editors to interact with the database? How would you protect database content from hackers and other unauthorized accesses?

RUNNING CASE STUDIES

Community Board of Realtors®

In Chapter 4, you developed a domain model class diagram. Using your previous solution or one provided to you by your instructor, update your domain model class diagram with any additional problem domain classes, new associations, or additional attributes that you have discovered as you worked through the previous chapters. Finalize this comprehensive domain model and then turn it in as part of your solution.

Using this comprehensive domain model class diagram, develop a relational database schema. In the schema, identify the foreign keys that are required.

Also, identify a key attribute for each table. You may need to add a key field if there isn't an attribute that could logically serve as the key. Remember that a candidate key for an association class is the combination of the keys of the connected classes. However, it may make sense to define a shorter, more concise key field.

Verify that each table is in first, second, and third normal form. Discuss any discrepancies you had to fix from your first solution. Discuss any tables that may not be in third normal form and why you are leaving it as non-normalized.

The Spring Breaks 'R' Us Travel Service

In Chapter 4, you developed a domain model class diagram. Using your previous solution or one provided to you by your instructor, update your domain model class diagram with any additional problem domain classes, new associations, or additional attributes that you have discovered as you worked toward your solutions in the previous chapters. Finalize this comprehensive domain model and then turn it in as part of your solution.

Using this comprehensive domain model class diagram, develop a relational database schema. In the schema, identify the foreign keys that are required. Also, identify a key attribute for each table. You may

need to add a key field if there isn't an attribute that could logically serve as the key. Remember that a candidate key for an association class is the combination of the keys of the connected classes. However, it may make sense to define a shorter, more concise key field.

Verify that each table is in first, second, and third normal form. Discuss any discrepancies you had to fix from your first solution. Discuss any tables that aren't in third normal form and why you are leaving them as non-normalized. (For example, in the United States, city and state are functionally dependent on zip code, but you might leave all three fields in the same table. Why?)

On the Spot Courier Services

In Chapter 4, you developed a domain model class diagram. Using your previous solution or one provided to you by your instructor, update your domain model class diagram with any additional problem domain classes, new associations, or additional attributes that you have discovered as you worked toward your solutions in the previous chapters. Finalize this comprehensive domain model and then turn it in as part of your solution.

Using this comprehensive domain model class diagram, develop a relational database schema. In the schema, identify the foreign keys that are required. Also, identify a key attribute for each table. You may need to add a key field if there isn't an attribute that could logically serve as the key. Remember that a candidate key for an association class is the combination

of the keys of the connected classes. However, it may make sense to define a shorter, more concise key field.

Verify that each table is in first, second, and third normal form. Discuss any discrepancies you had to fix from your first solution. Discuss any tables that aren't in third normal form and why you are leaving them as non-normalized.

Even though this is a small company, the DA and DBA responsibilities need to be assumed by somebody, or by several people. Of the employees so far identified in previous discussion, who should assume DA responsibilities and who should assume DBA responsibilities? Do the current employees have enough skills to successfully handle these responsibilities? Should "On the Spot" hire somebody? Explain your reasoning.

Sandia Medical Devices Real-Time Glucose Monitoring

Part 1

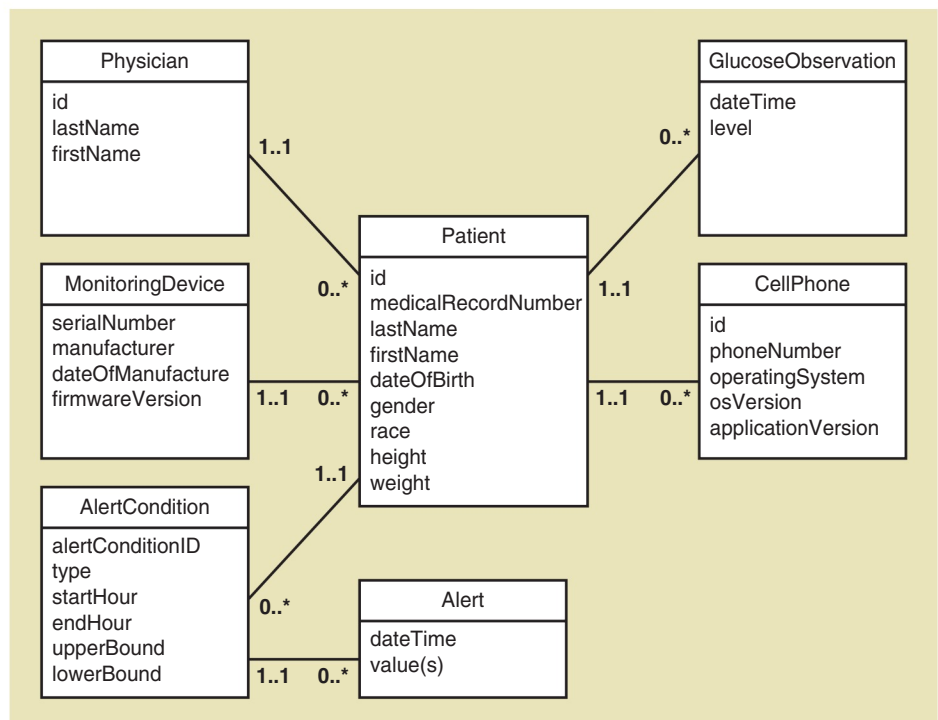
Review the original system description in Chapter 2, the additional project information in Chapters 3 and 4, and the domain class diagram shown in **Figure 9-29** to refamiliarize yourself with the proposed system. Assume that the type attribute of the AlertCondition class identifies one of three alert types:

1. Glucose levels that fall outside the specified range for 15 minutes (three consecutive readings)

2. Glucose levels that fall outside the specified range for 60 minutes (12 consecutive readings)
3. An average of glucose levels over an eight-hour period that falls outside a specified range

The specified range for an AlertCondition object is the set of values between and including lowerBound and upperBound. AlertCondition objects also include an effective time period specified by the attributes startHour and endHour, which enables physicians to set different alert parameters for sleeping and waking hours.

FIGURE 9-29 Updated domain model class diagram for Sandia RTGM system



When an alert is triggered, an object of type Alert is created and associated with an alertCondition object. The dateTime attribute records when the Alert object was created and the value(s) attribute records the glucose levels (alert types 1 and 2) or average level (alert type 3) that fell outside the specified range. Each Alert object is indirectly related to a Patient object via the association between Alert and AlertCondition and the association between AlertCondition and Patient.

Develop a set of relational database tables based on the domain class diagram. Identify all primary and foreign keys, and ensure that the tables are in 3NF.

Part 2

Based on what you learned in this chapter about databases, controls, and system security, review your answers to the questions for this case in Chapter 6. Assume that the patient's cell phone and the centralized servers are different nodes in a replicated database architecture and are regularly synchronized. What changes, if any, should be made to your answers now that you have a deeper understanding of databases, controls, security, and related design issues?

FURTHER RESOURCES

Alfred Basta and Melissa Zgola, *Database Security*. Cengage Learning, 2011.

Carlos Coronel, Steven Morris, and Peter Rob, *Database Systems: Design, Implementation, and Management* (9th ed.). Cengage Learning, 2010.

Michael E. Whitman and Herbert J. Mattord, *Principles of Information Security* (4th ed.). Cengage Learning, 2012.

D. David Arthur Bell and Jane B. Grimson, *Distributed Database Systems*. Addison-Wesley, 1992.



System Development and Project Management

PART FOUR

- **Chapter 10**
Approaches to System
Development
- **Chapter 11**
Project Planning and
Project Management
- **Online Chapter C**
Project Management Techniques



Approaches to System Development

CHAPTER TEN

CHAPTER OUTLINE

- The System Development Life Cycle
- Methodologies, Models, Tools, and Techniques
- Agile Development
- The Unified Process, Extreme Programming, and Scrum

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- Compare the underlying assumptions and uses of a predictive and an adaptive system development life cycle (SDLC)
- Explain what makes up a system development methodology—the SDLC as well as models, tools, and techniques
- Describe the key features of Agile development
- Understand and describe the key features of the Unified Process, Extreme Programming, and Scrum Agile system development methodologies

OPENING CASE DEVELOPMENT APPROACHES AT AJAX CORPORATION, CONSOLIDATED CONCEPTS, AND PINNACLE MANUFACTURING

Kim, Mary, and Bob, graduating seniors, were discussing their recent interview visits to companies that recruit computer information system (CIS) majors on their campus. All agreed that they had learned a lot by visiting the companies, but they also all felt somewhat overwhelmed.

"At first, I wasn't sure I knew what they were talking about," Kim said. During her on-campus interview, she had impressed Ajax Corporation with her knowledge of data modeling and database management. And when she visited the company's home office data center for her second interview, the interviewers spent a lot of time describing the company's system development methodology.

"They said to forget everything I learned in school," Kim continued. "That got my attention."

Ajax Corporation had purchased a complete development methodology called *IM One* from a small consulting firm. Most of its employees agreed that it worked fairly well, having invested a lot of time and money learning and adapting to it. Those who had worked for Ajax for a long time thought *IM One* was unique, and they were very proud of it.

"Then, they started telling me about their iterative SDLC, business events, use cases, and domain model class diagrams—things like that," Kim said. She had recognized that many of the key concepts in the *IM One* methodology were fairly standard models and techniques from UML and iterative development concepts that she had learned in school.

"I know what you mean," said Mary, a very talented programmer who knew just about every popular programming language available. "Consolidated Concepts went on and on about things like OMG, UP, and people named Booch, Rumbaugh, and Jacobson. It turns out they use

the object-oriented approach to develop their systems, and they like the fact that I know Java and C# and .NET. There was no problem once I got past all the people's names and acronyms they used. The actual models and techniques they used were very familiar to me."

Bob, who interviewed with Pinnacle Manufacturing, had a different story to tell. "A few people said analysis and design are no longer a big deal," he said. "And I'm thinking, 'Knowing that would have saved me some time in school.'"

Pinnacle has a small system development group supporting its manufacturing and inventory control. "They said they like to jump in and get to the code as soon as possible," Bob said. "Little documentation and not much of a project plan. They showed me some books on their desks, and it looked like they had been doing a lot of reading about analysis and design. I could see they were using Agile development and Agile modeling techniques and focusing on best practices required for their small projects. It turns out they just organize their work differently by looking at risk and writing user stories while building prototypes. I recognized some sketches of class diagrams and sequence diagrams on the boss's whiteboard, so I felt fairly comfortable that they were using an Agile approach like XP or Scrum but still doing some modeling after all."

Kim, Mary, and Bob agreed that there was much to learn in these work environments but also that there are many different ways to describe the key concepts and techniques they learned in school. They were all glad they focused on the fundamentals in their CIS classes and that they had been exposed to a variety of approaches to system development.

■ Overview

As the experiences of Kim, Mary, and Bob demonstrate, there are many ways to develop an information system, and doing so is very complex. Project managers rely on a variety of aids to help them with every step of the process. So far in this text, you have learned about analysis and design models and techniques, and now you will learn more about the overall system development process. You learned about the system development life cycle (SDLC) in Chapter 1. That particular SDLC included six core processes and multiple iterations. This chapter discusses the SDLC in more detail, including some variations found in industry.

The entire process of developing an information system requires more than just an SDLC. A system development methodology includes more specific instructions for completing the activities of each core process by using specific models, tools, and techniques. As mentioned in Chapter 1, a system development methodology is sometimes referred to as a development process.

Also as mentioned in Chapter 1, most newer system development methodologies are Agile. Agile development is further discussed in this chapter as a philosophy that guides a development project. Aside from being iterative, it focuses

on techniques and methods that encourage more user involvement and allow for more flexible projects with changing requirements. Finally, some specific Agile system development methodologies are described, including Agile Unified Process, Extreme Programming, and Scrum.

■ The System Development Life Cycle

Chapter 1 demonstrated how analysis and design models and techniques are used to solve business problems by building an information system. For problem-solving work to be productive, it needs to be organized and goal-oriented. Analysts achieve these results by organizing the work into projects. As defined in Chapter 1, a *project* is a planned undertaking, with a beginning and end that produces a well-defined result or product. The term *information system development project* refers to a planned undertaking that produces a new information system. Some system development projects are very large, requiring thousands of hours of work by many people and spanning several calendar years. In the RMO case study introduced in Chapter 2, the system being developed is a moderately sized computer-based information system requiring a moderately sized project. Many system development projects are smaller, lasting a few months. Some are very small, such as the Tradeshow application in Chapter 1.

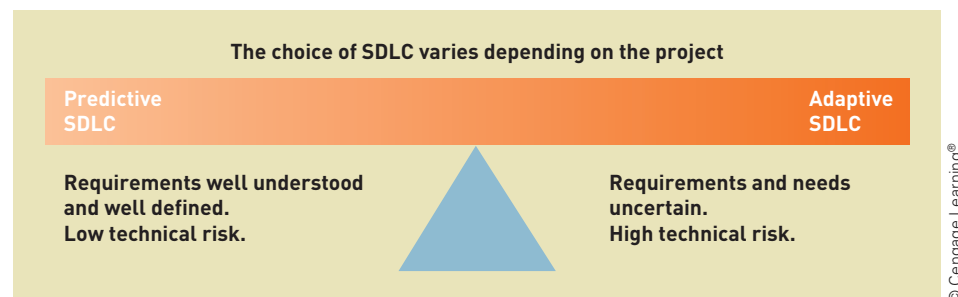
For a system development project to be successful, it must be planned and organized. The plan must include a comprehensive set of activities that flow in the proper sequence. Otherwise, activities are omitted or work may need to be done multiple times. The end result, of course, is producing a high-quality information system as measured by its reliability, robustness, efficiency, and fitness for purpose. The system development life cycle (SDLC), which was introduced in Chapter 1, is a fundamental concept in the success of information system development projects.

The SDLC provides a way to think about the development of a new system as a progressive process, much like a living entity. This concept can be expanded and you can view the information system as having a life itself; in fact, we often refer to the life cycle of a system. During its life cycle, an information system is first conceived, then it is designed, built, and deployed as part of a development project, and, finally, it is put into production and used to support the business. However, even during its productive use, a system is still a dynamic, living entity that is updated, modified, and repaired through smaller projects.

Several projects may be required during the life of a system, first to develop the original system and then to upgrade. This text focuses on the initial development project, not on the support projects. In other words, the primary concern is with getting the new system developed and deployed.

In today's diverse development environment, there are many approaches to developing systems, and they are based on different approaches to the SDLC. Although it is difficult to find a single, comprehensive classification system that encompasses all the approaches, one useful way to categorize them is along a continuum from predictive to adaptive (see **Figure 10-1**).

FIGURE 10-1 Predictive versus adaptive approaches to the SDLC



predictive approach to the SDLC an approach that assumes the project can be planned in advance and that the new information system can be developed according to the plan

adaptive approach to the SDLC an approach that assumes the project must be more flexible and adapt to changing needs as the project progresses

A **predictive approach to the SDLC** assumes that the development project can be planned and organized and that the new information system can be developed according to the plan. Predictive SDLCs are useful for building systems that are well understood and defined. For example, a company may want to convert its old networked client/server system to a newer Web-based system that includes a smartphone app. In this type of project, the staff already understands the requirements very well, and no new processes need to be added. Thus, the project can be carefully planned, and the system can be built according to the specifications.

An **adaptive approach to the SDLC** is used when the system's requirements and/or the users' needs aren't well understood. In this situation, the project can't be planned completely. Some system requirements may need to be determined after preliminary development work. Developers should still be able to build the solution, but they need to be flexible and adapt the project as it progresses. Recall that the Tradeshow System described in Chapter 1 used this approach.

In practice, any project could have—and most do have—predictive and adaptive elements. That is why Figure 10-1 shows them as endpoints along a continuum, not as mutually exclusive categories. The predictive approaches are more traditional and were conceived during the 1970s through the 1990s. Many of the newer, adaptive approaches have evolved with object-oriented technology and Web development; they were created during the late 1990s and into the twenty-first century. The following sections look at the more predictive approaches and then examine the newer adaptive approaches.

■ Traditional Predictive Approaches to the SDLC

The development of a new information system requires a number of different but related sets of activities. In predictive approaches, there is a group of activities that identifies the problem and secures approval to develop a new system; this is called *project initiation*. A second group of activities, called *project planning*, involves planning, organizing, and scheduling the project. These activities map out the project's overall structure. A third group—*analysis*—focuses on discovering and understanding the details of the problem or need. The intent here is to figure out exactly what the system must do to support the business processes. A fourth group—*design*—focuses on configuring and structuring the new system components. These activities use the requirements that were defined earlier to develop the program structure and the algorithms for the new system. A fifth group—*implementation*—includes programming and testing the system. A sixth group—*deployment*—involves installing and putting the system into operation.

These six groups of activities—project initiation, project planning, analysis, design, implementation, and deployment—are sometimes referred to as **phases** of the system development project, and they provide the framework for managing the project. Another phase, called the *support phase*, includes the activities needed to upgrade and maintain the system after it has been deployed. The support phase is part of the overall SDLC, but it isn't normally considered part of the initial development project. **Figure 10-2** illustrates the six phases of a

phases related groups of development activities, such as initiation, planning, analysis, design, implementation, deployment, and support

FIGURE 10-2 Traditional information system development phases (with support phase)

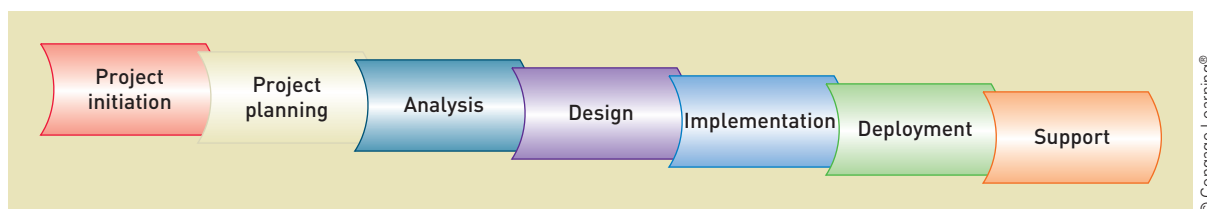
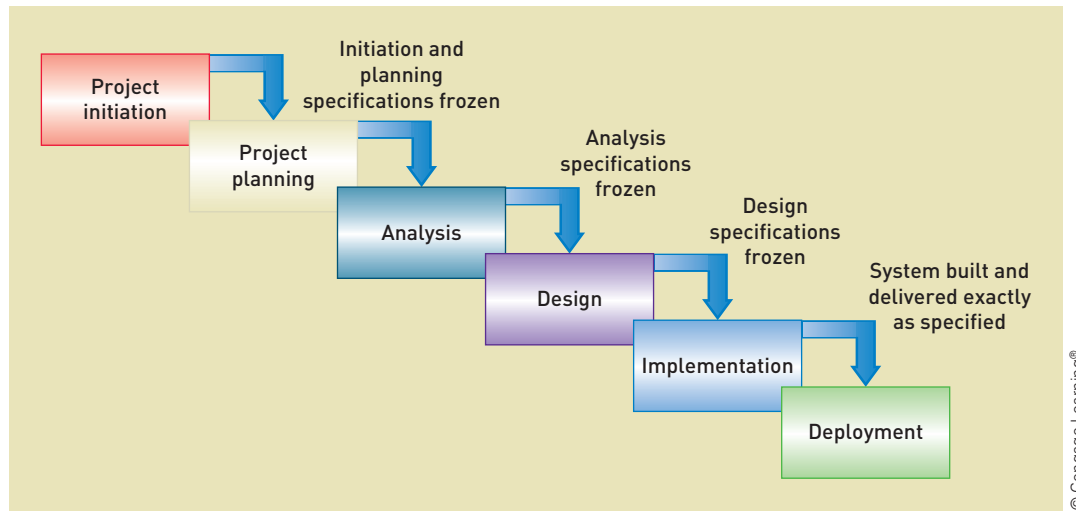


FIGURE 10-3 Waterfall model of the SDLC



waterfall model an SDLC approach that assumes the phases can be completed sequentially with no overlap

traditional predictive SDLC plus the support phase. Note that the main difference from the iterative SDLC used in this text is that the activities are carried out sequentially rather than repeated in each iteration.

The most predictive SDLC approach (i.e., farthest to the left on the predictive/adaptive scale) is called a **waterfall model**, with the phases of the project flowing down, one after another. As shown in **Figure 10-3**, this model assumes that the phases can be carried out and completed sequentially. First, a detailed plan is developed, then the requirements are thoroughly specified, then the system is designed down to the last algorithm, and then it is programmed, tested, and installed. After a project drops over the waterfall into the next phase, there is no going back. In practice, the waterfall model assumes rigid planning and final decision making at each step of the development project. As you may have guessed, the waterfall model doesn't always work very well. Being human, developers are rarely able to complete a phase without making mistakes or leaving out important components that have to be added later. However, even though the waterfall model isn't used in its purest form anymore, it still provides a valuable foundation for understanding development. No matter what system is being developed, you need to include initiation, planning, analysis, design, implementation, and deployment activities.

A little farther to the right on the predictive/adaptive scale are modified waterfall models. These are still predictive—that is, they still assume a fairly thorough plan—but there is a recognition that the project's phases must overlap, influencing and depending on each other. Some analysis must be done before the designing can start, but during the design, there is a need for more detail in the requirements or perhaps it is discovered that a requirement cannot be met in the manner originally requested. **Figure 10-4** illustrates how these activities can overlap.

■ Newer Adaptive Approaches to the SDLC

In an adaptive approach, project activities—including plans and models—are adjusted as the project progresses. There are many ways to depict an adaptive SDLC. All include iterations, which were discussed in Chapter 1. Rather than having the analysis, design, and implementation phases proceed sequentially with some overlap, iterations can be used to create a series of mini-projects that address smaller parts of the application. One of these smaller parts is analyzed, designed, built, and tested during a single iteration; then, based on the results, the next iteration proceeds to analyze, design, build, and test the next smaller part.

FIGURE 10-4 *Overlap of system development phases*

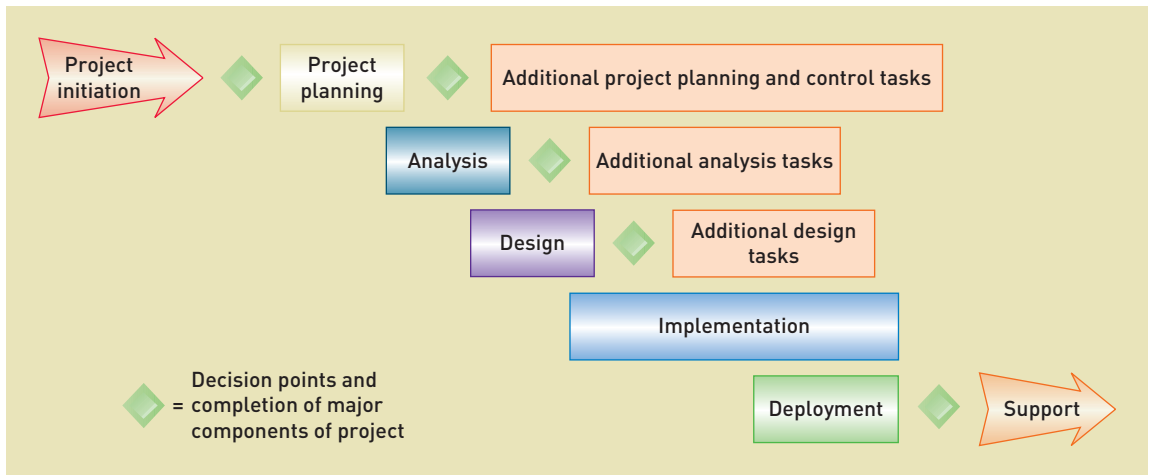


FIGURE 10-5 *Adaptive SDLC with six core processes and multiple iterations*

Core processes	Iterations					
	1	2	3	4	5	6
Identify the problem and obtain approval.	[Activity across all iterations]					
Plan and monitor project.	[Activity across all iterations]					
Discover and understand details.	[Activity across all iterations]					
Design system components.	[Activity across all iterations]					
Build, test, and integrate system components.	[Activity across all iterations]					
Complete system tests and deploy the solution.	[Activity across all iterations]					

Using iterations, the project is able to adapt to any changes as it proceeds. Also, parts of the system are available early on for user evaluation and feedback, which helps ensure that the application will meet the needs of the users.

You first saw this concept in the SDLC example used in Chapter 1, which is repeated here as **Figure 10-5**. The core processes defined in Chapter 1 are carried out in each iteration of the project. This iterative approach is adaptive because with each iteration’s analysis, design, and implementation, modifications can be made to adapt to the changing requirements of the project. The adaptive approach presented in this textbook is a simplification of and variation on a more formal iterative approach called the Unified Process (UP). You will learn more about the UP later in this chapter.

A related concept to an iterative SDLC is called **incremental development**. Incremental development is always based on an iterative life cycle. The basic idea is that the system is built in small increments. An increment may be developed within a single iteration or it may require two or three iterations. As each increment is completed, it is integrated with the whole. The system, in effect, is “grown” in an organic fashion. The advantage of this approach is that portions of the system get into the users’ hands much sooner so the business can begin accruing benefits as early as possible.

Yet another related concept, which is also based on an iterative approach, is the idea of a **walking skeleton**. A walking skeleton, as the name suggests, provides a complete front-to-back implementation of the new system but with only

incremental development an SDLC approach that completes portions of the system in small increments across iterations, with each increment being integrated into the whole as it is completed

walking skeleton a development approach in which the complete system structure is built but with bare-bones functionality

the “bare bones” of functionality. The walking skeleton is developed in a few iterations early in the project. Later iterations then flesh out the skeleton with more functions and capabilities. It should be obvious that this approach also gets working software into the hands of the users early in the project. Both these approaches have the additional advantage of extensive user testing and feedback to the project team as the project is progressing—another example of how an iterative project is also adaptive.

■ Methodologies, Models, Tools, and Techniques

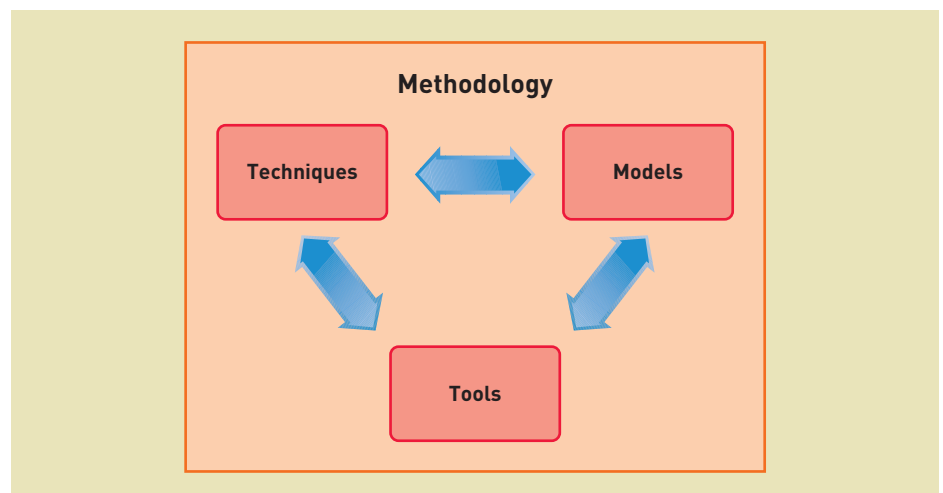
Aside from an SDLC, systems developers have a variety of aids at their disposal to help them complete activities and tasks. Among them are methodologies, models, tools, and techniques. The following sections discuss each of these aids.

■ Methodologies

A *system development methodology* as described in Chapter 1 provides guidelines for every facet of the system development life cycle. For example, within a methodology, certain models, such as diagrams, are used to describe and specify the requirements. Related to these models are the techniques for how the project team will do its work. An example of a technique is the guidelines for conducting a user interview that you learned about in Chapter 2. Finally, each project team will use a set of tools—usually computer-based tools—to build models, record information, and write the code. These tools are considered part of the overall methodology used for a given project. **Figure 10-6** illustrates that the techniques, models, and tools support one another to provide a comprehensive, integrated methodology. Some methodologies are developed by systems professionals within the company based on their experience. Others are learned and used based on purchased materials and training from consulting firms or vendors.

Some methodologies (whether created in-house or purchased) contain massive written documentation that defines everything the developers may need to produce at any point in the project, including how the documentation itself should look and what reports to management should contain. Other methodologies are much more informal, such as a single document that contains general descriptions of what needs to be done. Sometimes, the methodology a company adopts isn't only informal, it is ad hoc and almost undefined, but such freedom of choice is becoming rare. Management in most IT departments prefers to adopt a flexible methodology so it can be adapted to different types of projects and systems. The methodology used by the organization determines how predictive or adaptive the approach to a system development project should be.

FIGURE 10-6 Components of a methodology



■ Models

Anytime people need to record or communicate information, it is useful to create a model. As discussed in Chapter 2, a *model* is a representation of an important aspect of the real world. Sometimes, the term *abstraction* is used because people abstract (separate out) an aspect that is of particular importance to them. For example, consider an airplane model. To talk about the aerodynamics of the airplane, it is useful to have a model that shows the plane's overall shape in three dimensions. Sometimes, a drawing of the cross-sectional details of the plane's wing is what is needed. In other cases, a mathematical formulation of the plane's aerodynamic characteristics might be necessary to understand how it will behave.

Some models are physically similar to the real product. Some are graphical representations of important details. And some are abstract mathematical notations. Each emphasizes a particular type of information. In airplane design, engineers use lots of different models. Learning to be an aerospace engineer involves learning how to create and use all the various models. That is true for an information system developer too, although the models for information systems aren't as standardized or precise as aerospace models. System developers are making progress, but the field is very young, and many senior analysts were self-taught. More important, an information system is much less tangible than an airplane; you can't really see, hold, or touch it. Therefore, the models of the information system can seem much less tangible too.

The models used in system development include representations of inputs, outputs, processes, data, objects, object interactions, locations, networks, and devices, among other things. Most of the models are graphical models, which are drawn representations that employ agreed-upon symbols and conventions. These are often called diagrams and charts, and the UML diagrams you have encountered so far in this book are examples. Much of this text describes how to read and create a variety of models that represent an information system.

Another important kind of model is a project-planning model, such as a Gantt chart or net present value (NPV), both of which are discussed in Chapter 11. These models represent the system development project itself, highlighting its tasks and other considerations. Yet another model related to project management is a chart showing all the people assigned to the project. **Figure 10-7** lists some models used in system development.

FIGURE 10-7 Some models used in system development

Some models of system components

- Use case diagram
- Domain model class diagram
- Design class diagram
- Sequence diagram
- Package diagram
- Screen design template
- Dialog design storyboard
- Entity-relationship diagram (ERD)
- Database schema

Some models used to manage the development process

- Gantt chart
- Organizational hierarchy chart
- Financial analysis models—NPV, payback period
- System development life-cycle model
- Stakeholders list
- Iteration plan

tool a software application that assists developers in creating models or other components required for a project

integrated development environments (IDEs) sets of tools that work together to provide a comprehensive development and programming environment for software developers

visual modeling tools tools that help analysts create and verify graphical models and may also generate program code

technique a collection of guidelines that specify a method for how to carry out a development activity or task

■ Tools

In the context of system development, a **tool** is software support that helps create models or other components required in the project. Tools might be simple drawing programs for creating diagrams. They might also include an application that stores information about the project, such as data definitions, use case descriptions, and other artifacts. A project management software tool, such as Microsoft Project, is another example of a tool used to create models. The project management tool creates a model of the project tasks and task dependencies.

Tools have been specifically designed to help system developers. Programmers should be familiar with **integrated development environments (IDEs)**, which include many tools to help with programming tasks—for example, smart editors, context-sensitive help, and debugging tools. Some tools can generate program code for the developer. Some tools reverse engineer old programs, generating a model from the code so the developer can determine what the program does if its documentation is missing (or was never done). **Visual modeling tools** are available to systems analysts to help them create and verify important system models. These tools are used to draw such diagrams as class diagrams or activity diagrams. Other visual modeling tools help design the database or even generate program code. **Figure 10-8** lists the types of tools used in system development.

■ Techniques

You learned several techniques for gathering information in Chapter 2. You learned many techniques for defining functional requirements in Chapters 3, 4, and 5, and many design techniques in Chapters 6, 7, 8, and 9. In system development, a **technique** is a collection of guidelines that helps an analyst complete an activity or task. It often includes step-by-step instructions for creating a model, or it might include more general advice on collecting information from system users. Examples include data-modeling techniques, software-testing techniques, user-interviewing techniques, and relational database design techniques.

Sometimes, a technique applies to an entire life cycle phase and helps you create several models and other documents. The modern structured analysis technique (discussed later) is an example of this. **Figure 10-9** lists some techniques commonly used in system development.

How do methodologies, models, tools, and techniques fit together? A *methodology* includes a collection of *techniques* that are used to complete activities within each phase or iteration of the system development life cycle. The activities include the completion of a variety of *models* as well as other documents and deliverables. Like any other professionals, system developers use software *tools* to help them complete their activities.

FIGURE 10-8 Types of tools used in system development

- Project management application
- Drawing/graphics application
- Word processor/text editor
- Visual modeling tool
- Integrated development environment (IDE)
- Database management application
- Reverse-engineering tool
- Code generator tool

FIGURE 10-9 Some techniques used in system development

Strategic planning techniques
 Project management techniques
 User-interviewing techniques
 Data-modeling techniques
 Relational database design techniques
 Structured programming techniques
 Software-testing techniques
 Process modeling techniques
 Domain modeling techniques
 Use case modeling techniques
 Object-oriented programming techniques
 Architectural design techniques
 User-interface design techniques

© Cengage Learning®

■ Agile Development

Agile development a guiding philosophy and set of guidelines for developing information systems in an unknown, rapidly changing environment

The highly volatile marketplace has forced businesses to respond rapidly to new opportunities. Sometimes, these opportunities appear in the middle of implementing another business initiative. To survive, businesses must be agile—that is, able to change directions rapidly, even in the middle of a project. **Agile development** is a philosophy and set of guidelines for developing information systems in an unknown, rapidly changing environment, and it can be used with any system development methodology. Usually, Agile development complements adaptive approaches to the SDLC and methodologies that support it. But the emphasis is on taking an adaptive approach and making it agile in all development activities and tasks. Related to Agile development, Agile modeling is a philosophy about how to build models, some of which are formal and detailed, others sketchy and minimal. All the models you have learned how to create in this text can be used with Agile modeling.

■ Agile Development Philosophy and Values

The “Manifesto for Agile Software Development” (see the “Further Resources” section) identifies four basic values, which represent the core philosophy of Agile development:

- Value responding to change over following a plan
- Value individuals and interactions over processes and tools
- Value working software over comprehensive documentation
- Value customer collaboration over contract negotiation

The people involved in system development—whether as team members, users, or other stakeholders—all need to accept these priorities for a project to be truly agile. Adopting an Agile approach isn’t always easy. Managers and executive stakeholders frequently have trouble accepting this less-rigid viewpoint, wanting instead to impose more controls on development teams and enforce detailed plans and schedules. However, the Agile philosophy takes the opposite approach, providing more flexibility in project schedules and letting the project teams plan and execute their work as the project progresses.

chaordic a term used to describe adaptive projects because they are chaotic yet ordered

Some industry leaders in the Agile movement coined the term **chaordic** to describe an Agile project. *Chaordic* comes from two words: *chaos* and *order*. The first two values in the list do seem to be a recipe for chaos, but software projects always have unpredictable elements—hence, a certain amount of chaos. The Agile philosophy recognizes this unpredictability, handling it with increased flexibility and by trusting the project team to develop solutions to project problems. Depending too heavily on a plan and predefined processes

exacerbates problems when unpredictable requirements arise. Developers need to accept a certain amount of chaos and mix that with other Agile modeling and development techniques that help to provide order and structure to the project. Chapter 11 will cover many of these Agile project management techniques.

Another important aspect of Agile development is that customers must continually be involved with the project team. They don't just sit down with the project team for a few sessions to develop the specifications and then go their separate ways. They become part of the technical team. Because working software is being developed throughout the project, customers are continually involved in defining requirements and testing components.

Historically, particularly with predictive projects, many system development efforts attempted to be fixed-price endeavors. This was true for both in-house groups and external development teams. However, the approach with Agile development is that system development projects should be more of a collaborative effort. Hence, contracts take on an entirely different flavor. Fixed prices and fixed deliverables don't make sense. Contracts for Agile projects include other kinds of options for the customer. The approach to the scheduling of activities, the delivery of system components, and the early termination of the project allow the client to maintain control, but it is done with different options than with fixed-bid contracts.

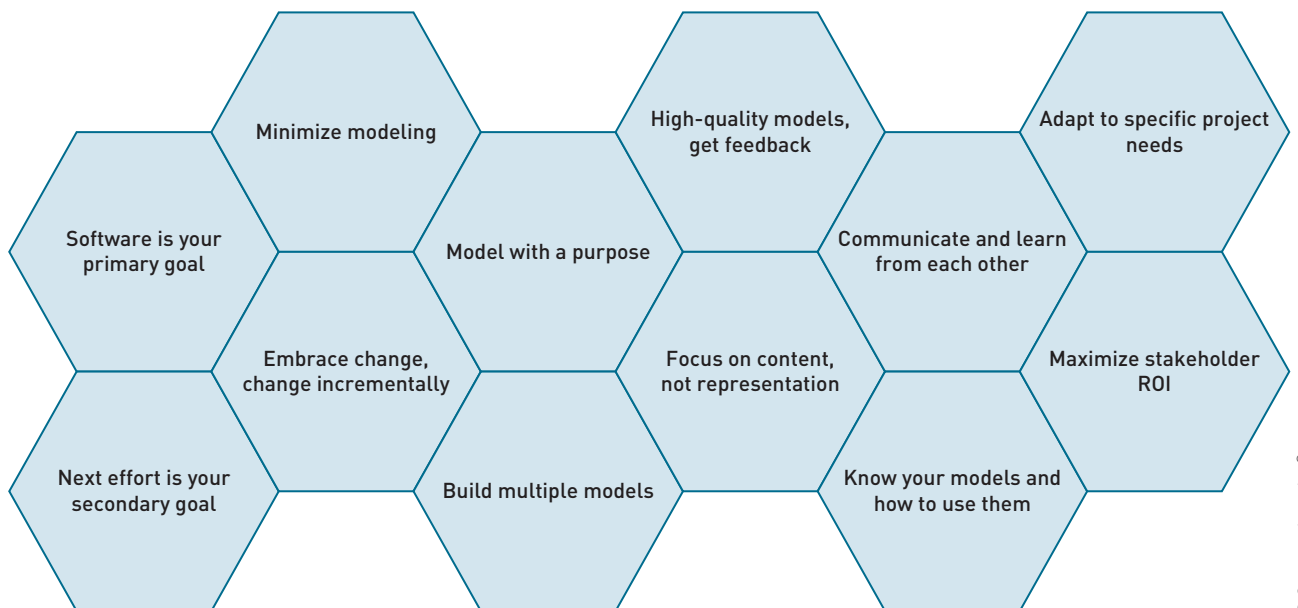
Models and modeling are critical to Agile development, so the next section looks at Agile modeling. Many of the core values are illustrated in the principles and practices of building models.

■ Agile Modeling Principles

Much of this text teaches techniques for creating models. Your first impression might be that an Agile approach means less modeling or maybe even no modeling. **Agile modeling (AM)** isn't about doing less modeling, but about doing the right kind of modeling at the right level of detail for the right purposes. AM doesn't dictate which models to build or how formal to make those models. Instead, it helps developers stay on track with their models by using them as a means to an end rather than end deliverables. AM's basic principles express the attitude that developers should have as they develop software. **Figure 10-10** summarizes Agile modeling principles. Those principles are discussed next.

Agile modeling (AM) a guiding philosophy in which only models that are necessary, with a valid need and at the right level of detail, are created

FIGURE 10-10 Agile modeling principles



■ Develop Software as Your Primary Goal

The primary goal of a software development project is to produce high-quality software. The primary measurement of progress is working software, not intermediate models of system requirements or specifications. Modeling is always a means to an end, not the end itself. Any activity that doesn't directly contribute to the end goal of producing software should be questioned and avoided if it cannot be justified.

■ Enable the Next Effort as Your Secondary Goal

Focusing only on working software can also be self-defeating, so developers must consider two important objectives. First, requirements models might be necessary to develop design models. So, don't think that if the model cannot be used to write code, it is unnecessary. Sometimes, several intermediate steps are needed before the final code can be written. Second, although high-quality software is the primary goal, long-term use of that code is also important. So, some models might be necessary to support maintenance and enhancement of the system. Yes, the code is the best documentation, but some architectural design decisions might not be easily identified from the code. Look carefully at what other artifacts might be necessary to produce high-quality systems in the long term.

■ Minimize Your Modeling Activity—Few and Simple

Create only the models that are necessary. Do just enough to get by. This principle isn't a justification for sloppy work or inadequate analysis. The models you create should be clear, correct, and complete. But don't create unnecessary models. Also, keep each model as simple as possible. Normally, the simplest solution is the best solution. Elaborate solutions tend to be difficult to understand and maintain. However, simplicity isn't a justification for being incomplete.

■ Embrace Change and Change Incrementally

Because the underlying philosophy of Agile modeling is that developers must be flexible and respond quickly to change, a good Agile developer willingly accepts—even embraces—change. Change is seen as the norm, not the exception. Watch for change, and have procedures ready to integrate changes into the models. The best way to accept change is to develop incrementally. Take small steps and address problems in small bites. Change your model incrementally and then validate it to make sure it is correct. Don't try to accomplish everything in one big release.

■ Model with a Purpose

Recall that the two reasons to build models are to understand what you are building and to communicate important aspects of the solution system. Make sure your modeling efforts support those reasons. Sometimes, developers try to justify building models by claiming that (1) the development methodology mandates the development of the model, (2) someone wants a model, even though the person doesn't know why it is important, or (3) a model can replace a face-to-face discussion of issues. Identify a reason and an audience for each model you develop. Then, develop the model in sufficient detail to satisfy the reason and the audience. Incidentally, the audience might be you.

■ Build Multiple Models

Along with other modeling methodologies, UML has several models to represent different aspects of the problem at hand. To be successful—in understanding the problem or communicating the solution—you need to model the critical aspects of the problem domain or the required solution. Don't develop all of them; be sure to minimize your modeling, but develop enough models to make sure you have addressed all the issues.

■ Build High-Quality Models and Get Feedback Rapidly

Nobody likes sloppy work. It is based on faulty thinking and introduces errors. One way to avoid error in models is to get feedback rapidly while the work is still fresh. Feedback comes from users as well as from technical team members. Others will have helpful insights and different ways to view a problem and identify a solution.

■ Focus on Content Rather Than Representation

Sometimes, a project team has access to a sophisticated visual modeling tool. These can be helpful, but at times, they are distracting because developers spend time making the diagrams pretty. Be judicious in the use of tools. Some models need to be well drawn for communication or contractual issues. Sometimes, it is more productive to build a model with a tool because it is expected that it will be changed frequently, and using a tool is usually more productive than redrawing by hand. In other cases, a hand-drawn diagram might suffice. Sometimes, developers work out a model on a whiteboard in a conference room and take a digital photo to keep a record of the details worked out.

■ Learn from Each Other with Open Communication

All the adaptive approaches emphasize working in teams. Don't be defensive about your models. Other team members have good suggestions. You can never truly master every aspect of a problem or its models.

■ Know Your Models and How to Use Them

Being an Agile modeler doesn't mean that you aren't skilled. If anything, you must be *more* skilled to know the strengths and weaknesses of the models, including how and when to use them. An expert modeler applies the previous principles of simplicity, quality, and development of multiple models.

■ Adapt to Specific Project Needs

Every project is different because it exists in a unique environment; involves different users, stakeholders, and team members; and requires a different development environment and deployment platform. Adapt your models and modeling techniques to fit the needs of the business and the project. Sometimes, models can be informal and simple. For other projects, more formal, complicated models might be required. An Agile modeler is able to adapt to each project.

■ Maximize Stakeholder ROI

The stakeholders are the real purpose that there is even a project. Stakeholders include both the users of the new system and those who are funding the development of the system. Never lose track of the real reason for the project. The stakeholders deserve to have the final say in what the system does and how it is developed.

■ The Unified Process, Extreme Programming, and Scrum

The Agile philosophy has proven to be an effective way to approach software development in today's fast-paced, continually changing landscape of computer applications. However, the Agile philosophy only proposes principles; it isn't meant to be a complete methodology, with practices and action steps. This section presents three methodologies that incorporate Agile principles, but are also complete methodologies, with specific techniques and practices.

These three methodologies—UP, XP, and Scrum—are among the most popular approaches to application software development, but they aren't always

found in their purest forms. Frequently, organizations either mix and match techniques from the three or only adopt a specific set of practices. However, adoption of these methodologies continues to expand throughout all types of organizations that develop software applications.

There are other variations of Agile development that are also popular. Each has its own adherents and its own set of principles and practices. Some of these other methodologies include Lean Software Development, Kanban Development, and Feature Driven Development (FDD). As you move into your career in software development, you may have opportunities to learn and practice several variations of Agile development methodologies.

■ The Unified Process

The Unified Process (UP) is an object-oriented system development methodology originally offered by Rational Software, which is now part of IBM. Developed by Grady Booch, James Rumbaugh, and Ivar Jacobson—the three pioneers behind the success of the Unified Modeling Language (UML)—the UP defines a complete methodology that uses UML for system models and describes a new, adaptive system development life cycle. In the UP, the term *development process* is synonymous with development methodology.

The UP is now widely recognized as a highly influential innovation in software development methodologies for object-oriented development using an adaptive approach. The original version of UP defined an elaborate set of activities and deliverables for every step of the development process. More recent versions are streamlined, with fewer activities and deliverables, simplifying the methodology. The methodology used in this textbook is an adaptation of UP principles.

As discussed previously, adaptive methodologies—including the UP—are all based on an iterative approach to development. You learned in Chapter 1 that each iteration is like a mini-project, in which requirements are defined based on analysis tasks, system components are designed, and those components are then implemented—at least partially—through programming and testing. However, one of the big questions in adaptive development is what the focus of each iteration should be. In other words, do iterations early in the project have the same objectives and focus as those done later? The UP answers this question by dividing a project into four major phases.

■ Up Phases

A phase in the UP can be thought of as a goal or major emphasis for a particular portion of the project. The four phases of the UP life cycle are inception, elaboration, construction, and transition, as shown in **Figure 10-11**.

Each phase of the UP life cycle describes the emphasis or objectives of the project team members and their activities at that point in time. Thus, the four phases provide a general framework for planning and tracking the project over time. Within each phase, several iterations are planned to give the team enough flexibility to adjust to problems or changing conditions. The emphases or objectives of the project team in each of the four phases are described briefly in **Figure 10-12**.

Inception Phase As in any project-planning phase, the inception phase consists of the project manager developing and refining a vision for the new system in order

FIGURE 10-11 The Unified Process system development life cycle with phases and iterations

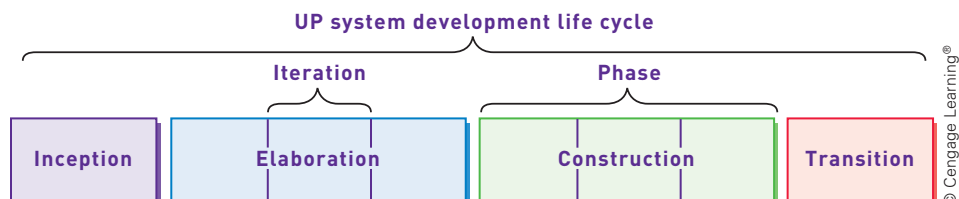


FIGURE 10-12 UP phases and their objectives

UP Phase	Objective
Inception	Develop an approximate vision of the system, make the business case, define the scope, and produce rough estimates for cost and schedule.
Elaboration	Define the vision, identify and describe all requirements, finalize the scope, design and implement the core architecture and functions, resolve high risks, and produce realistic estimates for cost and schedule.
Construction	Iteratively implement the remaining lower-risk, predictable, and easier elements and prepare for deployment.
Transition	Complete the beta test and deployment so users have a working system and are ready to benefit as expected.

© Cengage Learning®

to show how it will improve operations and solve existing problems. Essentially, the project manager makes the business case for the new system, proving that the new system's benefits will outweigh the cost of development. The scope of the system must also be defined so it is clear what the project will accomplish. Defining the scope includes identifying many of the key requirements for the system.

The inception phase is usually completed in one iteration, and as with any iteration, parts of the actual system may be designed, implemented, and tested. As software is developed, team members must confirm that the system vision still matches user expectations or that the technology will work as planned. Sometimes, prototypes are discarded after proving that point.

Elaboration Phase The elaboration phase usually involves several iterations, and early iterations typically complete the identification and definition of all the system requirements. Because the UP is an adaptive approach to development, the requirements are expected to evolve and change after work starts on the project.

The elaboration phase's iterations also complete the analysis, design, and implementation of the system's core architecture. Usually, the aspects of the system that pose the greatest risk are identified and implemented first. Until developers know exactly how the highest-risk aspects of the project will work out, they can't determine the amount of effort required to complete the project. By the end of the elaboration phase, the project manager should have more realistic estimates for the project's cost and schedule, and the business case for the project can be confirmed. Remember that the design, implementation, and testing of key parts of the system are completed during the elaboration phase. One other major objective of the elaboration phase is to do the necessary research and fact-finding so all the user requirements are identified. During the elaboration phase, a high percentage of time is spent on understanding and analysis.

Construction Phase The construction phase involves several iterations that continue the design and implementation of the system. The core architecture and highest-risk aspects of the system are already complete. Now the focus of the work turns to the routine and predictable parts of the system—for example, detailing the system controls, such as data validation, fine-tuning the user-interface design, finishing routine data maintenance functions, and completing the help and user preference functions. The team also begins to plan for deployment of the system.

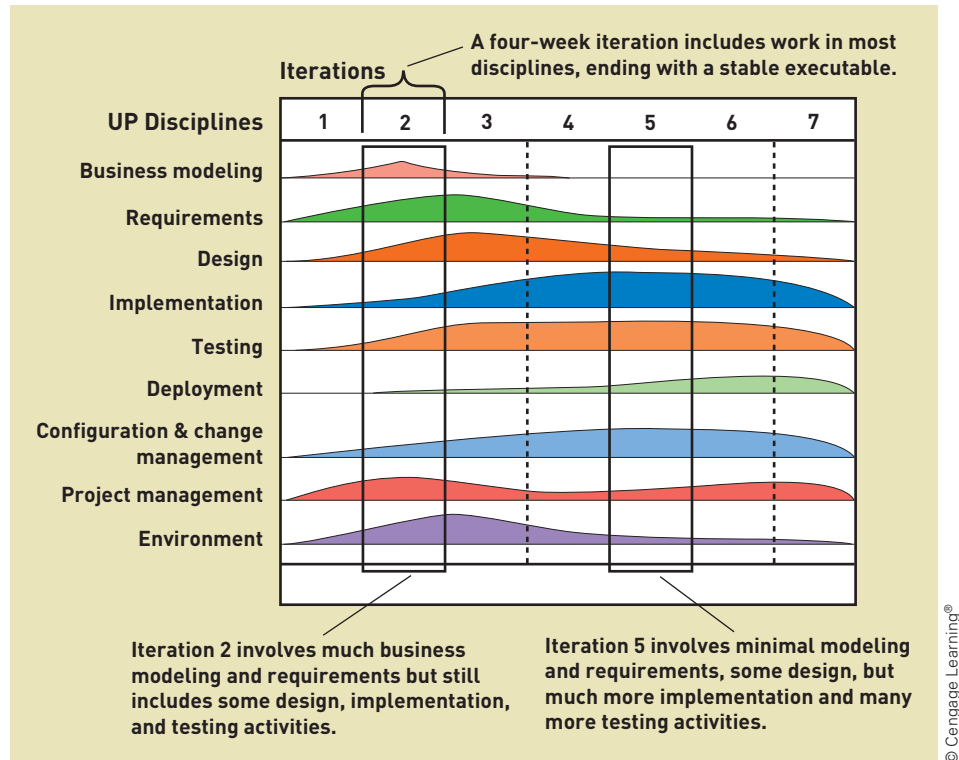
Transition Phase During the transition phase, one or more final iterations involve the final user acceptance and beta tests, and the system is made ready for operation. After the system is in operation, it will need to be supported and maintained.

■ UP Disciplines

As mentioned earlier, the four UP phases define the project sequentially by indicating the emphasis of the project team at any point in time. To make iterative development manageable, the UP defines disciplines to use within each iteration. A **UP discipline** is a set of functionally related activities that contributes to

UP discipline a set of functionally related activities that combine to enable the development process in a UP project

FIGURE 10-13 UP disciplines used in varying amounts in each iteration



one aspect of the development project. UP disciplines include business modeling, requirements, design, implementation, testing, deployment, configuration and change management, project management, and environment. Each iteration usually involves activities from all disciplines.

Figure 10-13 shows how the UP disciplines are involved in each iteration, which is typically planned to last four weeks. The size of the shaded area under the curve for each discipline indicates the relative amount of work included from each discipline during the iteration. This is the same approach to showing work done in each iteration as the SDLC used throughout this text. The amount and nature of the work differs from iteration to iteration. For example, in iteration 2, much of the effort focuses on business modeling and requirements definition, with much less effort focused on implementation and deployment. In iteration 5, very little effort is focused on modeling and requirements and much more effort focused on implementation, testing, and deployment. But most iterations involve some work in all disciplines.

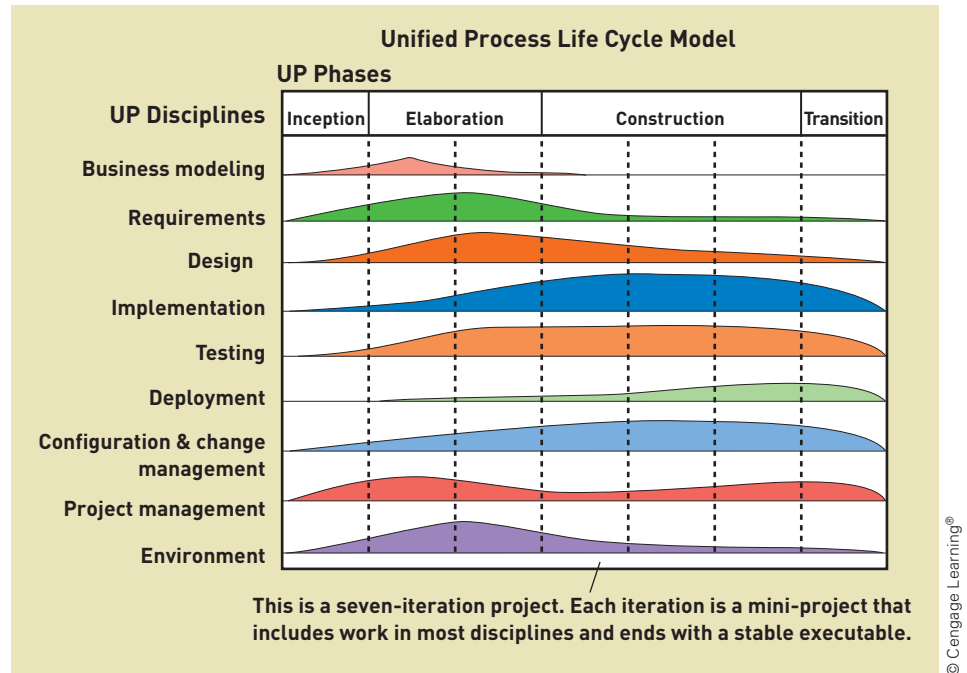
Figure 10-14 shows the entire UP life cycle: phases, iterations, and disciplines. It includes all the key UP life cycle features and is useful for understanding how a typical information system development project is managed.

The previous figures illustrate how the phases include activities from each discipline. But what about the detailed activities that occur within each discipline? The disciplines can be divided into two main categories: system development activities and project management activities. The six main UP development disciplines are:

- Business modeling
- Requirements
- Design
- Implementation
- Testing
- Deployment

For each iteration, the project team must understand the business environment (business modeling), define the requirements that that portion of the system must satisfy (requirements), design a solution for that portion of the system

FIGURE 10-14 UP life cycle with phases, iterations, and disciplines



that satisfies the requirements (design), write and integrate the computer code that makes that portion of the system work (implementation), thoroughly test that portion of the system (testing), and then, in some cases, put the part of the system that is completed and tested into operation for users (deployment).

Three additional support disciplines are necessary for planning and controlling the project:

- Configuration and change management
- Project management
- Environment

Configuration and change management involves setting up processes to support the coding activities. This includes such guidelines as when and how to release code as well as when and how to manage releases and versions. Project management refers to the tasks that are discussed in Chapter 11, such as planning the iterations, assigning work, and verifying that work has been completed. The environment discipline involves those tasks required to establish the working environment, including the tools to be used by the team. It also includes those guidelines about how to work together in an iterative Agile project.

All nine UP disciplines are employed throughout the lifetime of a project but to different degrees. For example, in the inception phase, there is one iteration. During the inception phase iteration, the project manager might complete a model showing some aspect of the system environment (the business modeling discipline). The scope of the system is delineated by defining many of the key system requirements and listing use cases (the requirements discipline). To prove technological feasibility, some technical aspect of the system might be designed (the design discipline), programmed (the implementation discipline), and tested to make sure it will work as planned (the testing discipline). In addition, the project manager makes plans for handling changes to the project (the configuration and change management discipline), working on a schedule and cost/benefit analysis (the project management discipline), and tailoring the UP phases, iterations, deliverables, and tools to match the needs of the project (the environment discipline).

The elaboration phase includes several iterations. In the first iteration, the team works on the details of the domain classes and use cases addressed in the iteration (the business modeling and requirements disciplines). At the same

time, it might complete the description of all use cases to finalize the scope (the requirements discipline). The use cases addressed in the iteration are designed by creating design class diagrams and interaction diagrams (the design discipline), programmed using Java or Visual Basic .NET (the implementation discipline), and fully tested (the testing discipline). The project manager works on the plan for the next iteration and continues to refine the schedule and feasibility assessments (the project management discipline), and all team members continue to receive training on the UP activities they are completing and the system development tools they are using (the environment discipline).

By the time the project progresses to the construction phase, most of the use cases have been designed and implemented in their initial form. The focus of the project turns to satisfying other technical, performance, and reliability requirements for each use case, finalizing the design, and implementing the design. These requirements are usually routine and lower risk, but they are key to the success of the system. The effort focuses on designing system controls and security and on implementing and testing these aspects.

As a system development methodology, the Unified Process must be tailored to the development team and the specific project. Choices must be made about which deliverables to produce and the level of formality, or ceremony, to be used. Sometimes, a project requires formal reporting and controls. Other times, it can be less formal. The UP should always be tailored to the project, although the UP does tend toward more ceremony than the next two methodologies.

■ Extreme Programming

Extreme Programming (XP) is an adaptive, Agile development methodology that was created in the mid-1990s. The word *extreme* sometimes makes people think that this methodology is completely new and that developers who embrace XP are radicals. However, XP is really an attempt to take the best practices of software development and extend them “to the extreme.” Extreme programming has these characteristics:

- Takes proven industry best practices and focuses on them intensely
- Combines those best practices (in their most intense forms) in a new way to produce a result that is greater than the sum of its parts

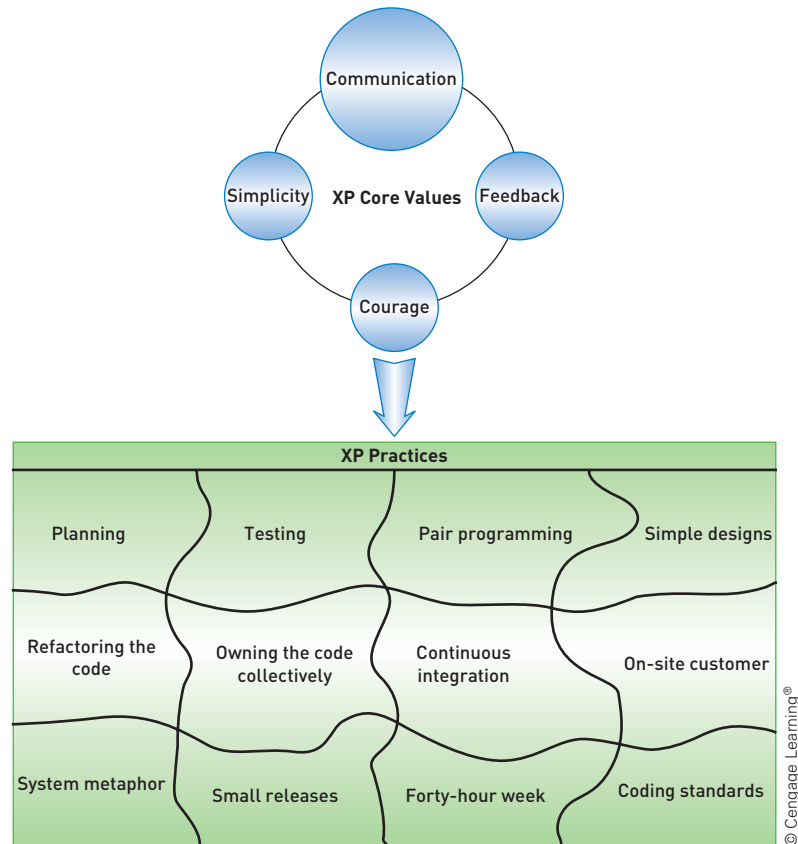
Figure 10-15 lists the core values and practices of XP. The following sections first present the four core values of XP and then explain its 12 primary practices. Finally, the basic structure of an XP project and the way XP is used to develop software is discussed.

■ XP Core Values

The four core values of XP—communication, simplicity, feedback, and courage—drive its practices and project activities. You will recognize the first three as best practices for any development project. You will also notice that the fourth is a desired value for any project, even though it might not be stated explicitly. Here are brief descriptions of the four core values of XP:

- **Communication.** One of the major causes of project failure is a lack of open communication among the right players at the right time and at the right level. Effective communication involves not only documentation, but also verbal discussion. The practices and methods of XP are designed to ensure that open, frequent communication occurs.
- **Simplicity.** Even though developers have always advocated keeping solutions simple, they don’t always follow their own advice. XP includes techniques to reinforce this principle and make it a standard way of developing systems.
- **Feedback.** As with simplicity, getting frequent, meaningful feedback is recognized as a best practice of software development. Feedback on functionality and requirements should come from the users, feedback on designs

FIGURE 10-15 XP core values and practices



and code should come from other developers, and feedback on satisfying a business need should come from the client. XP integrates feedback into every aspect of development.

- **Courage.** Developers always need courage to face the harsh choice of doing things right or throwing away bad code and starting over. But all too frequently, they haven't had the courage to stand up to a too-tight schedule, resulting in bad mistakes. XP practices are designed to give developers the courage to "do it right."

■ XP Practices

XP's 12 practices embody the basic values just presented. These practices are consistent with the Agile principles explained earlier in this chapter.

Planning Some people describe XP as glorified hacking or as the old "code and fix" methodology that was used in the 1960s. That isn't true; XP does include planning. However, as an adaptive technique, it recognizes that you can't know everything at the start. As indicated earlier, XP embraces change. XP planning focuses on making a rough plan quickly and then refining it as things become clearer. This reflects the Agile development philosophical dictum that change is more important than detailed plans. It is also consistent with the idea that individuals—and their abilities—are more important than an elaborate process.

The basis of an XP plan is a set of stories that users develop. A story describes what the system needs to do. XP doesn't use the term *use case*, but a user story and a use case express a similar idea. Planning involves two aspects: business issues and technical issues. In XP, the business issues are decided by the users and clients, whereas technical issues are decided by the development team. The plan, especially in the early stages of the project, consists of the list of stories (from the users) and the estimates of effort, risk, and work dependencies for each story (from the development team). As in Agile development, the idea is to heavily involve the users in the project rather than have them to simply sign off on specifications.

pair programming XP practice in which two programmers work together on designing, coding, and testing software

refactoring revising, reorganizing, and rebuilding part of a system so it is of higher quality

Testing Every new piece of software requires testing, and every methodology includes testing. XP intensifies testing by requiring that the tests for each story be written first—before the solution is programmed. There are two major types of tests: unit tests, which test the correctness of a small piece of code, and acceptance tests, which test the business function. The developers write the unit tests, and the users write the acceptance tests. Before any code can be integrated into the library of the growing system, it must pass the tests. By having the tests written first, XP automates their use and executes them frequently. Over time, a library of required tests is created, so when requirements change and the code needs to be updated, the tests can be rerun quickly and automatically.

Pair Programming More than any other, this practice is one for which XP is famous. Instead of simply requiring one programmer to watch another’s work, **pair programming** divides up the coding work. First, one programmer might focus more on design and double-checking the algorithms while the other writes the code. Then, they switch roles; thus, over time, they both think about design, coding, and testing. XP relies on comprehensive and continual code reviews. Interestingly, research has shown that pair programming is more efficient than programming alone. It takes longer to write the initial code, but the long-term quality is higher. Errors are caught quickly and early, two people become familiar with every part of the system, all design decisions are developed by two brains, and fewer “quick and dirty” shortcuts are taken. The quality of the code is always higher in a pair-programming environment.

Simple Designs Opponents say that XP neglects design, but that isn’t true. XP conforms to the principles of Agile modeling by avoiding the “Big Design Up Front” approach. Instead, it views design as so important that it should be done continually, although in small chunks. As with everything else, the design must be verified immediately by reviewing it along with coding and testing.

So, what is a simple design? It is one that accomplishes the desired result with as few classes and methods as possible and that doesn’t duplicate code. Accomplishing all that is often a major challenge.

Refactoring the Code **Refactoring** is the technique of improving the code without changing what it does. XP programmers continually refactor their code. Before and after adding any new functions, XP programmers review their code to see whether there is a simpler design or a simpler method of achieving the same result. Refactoring produces high-quality, robust code.

Owning the Code Collectively In XP, everyone is responsible for the code. No one person can say, “This is my code.” Someone can say, “I wrote it,” but everyone owns it. Collective ownership allows anyone to modify any piece of code. However, because unit tests are run before and after every change, if programmers see something that needs fixing, they can run the unit tests to make sure the change didn’t break something. This practice embodies the team concept that developers are building a system together.

Continuous Integration This practice embodies XP’s idea of “growing” the software. Small pieces of code—which have passed the unit tests—are integrated into the system daily or even more often. Continuous integration highlights errors rapidly and keeps the project moving ahead. The traditional approach of integrating large chunks of code late in the project often resulted in tremendous amounts of rework and time lost while developers tried to determine just what went wrong. XP’s practice of continuous integration prevents that.

On-Site Customer As with all adaptive approaches, XP projects require continual involvement of users who can make business decisions about functionality and scope. Based on the core value of communication, this practice keeps the project moving ahead rapidly. If the customer isn’t ready to commit resources to the project, the project won’t be very successful.

System Metaphor This practice is XP’s unique and interesting approach to defining an architectural vision. It answers the questions “How does the system work?” and “What are its major components?” And it does it by having the developers identify a metaphor for the system. For example, Big Three automaker Chrysler’s payroll system was built as a production-line metaphor, with its system components using production-line terms. Everyone at Chrysler understood a production line, so a payroll transaction was treated the same way; developers started with a basic transaction and then applied various processes to complete it. Of course, a system metaphor should be easily understood or well known to the members of the development team. It can guide members toward a vision and help them understand the system.

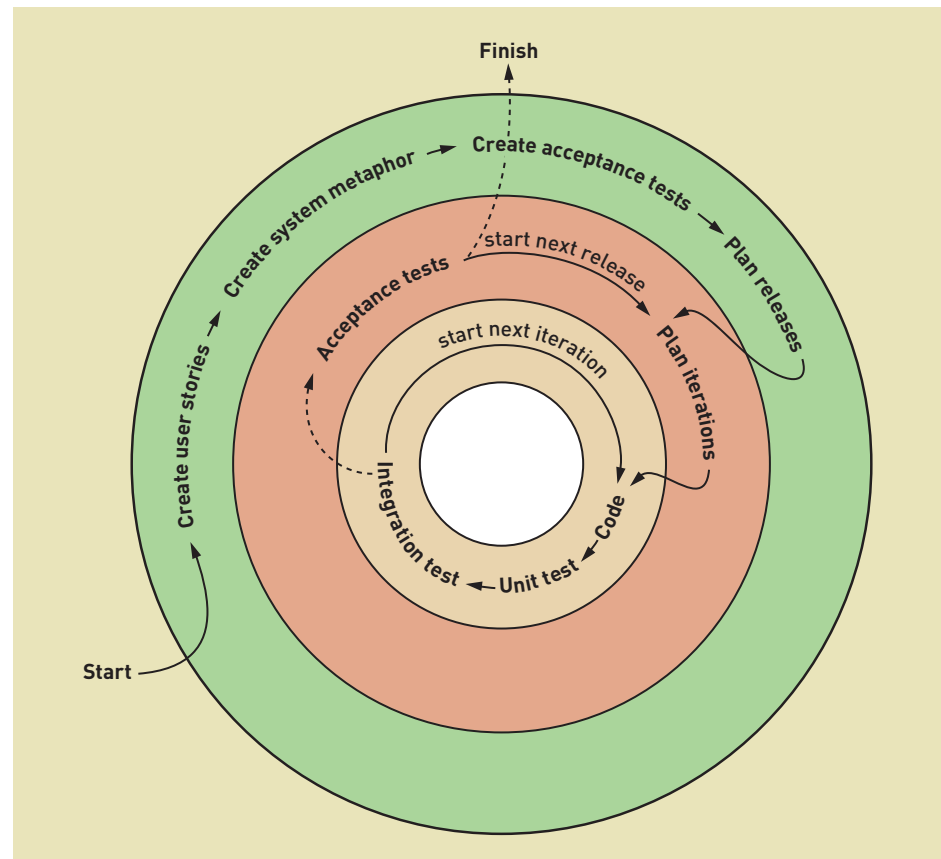
Small Releases A release is a point at which the new system can be turned over to users for acceptance testing and even for productive use. Consistent with the entire philosophy of growing the software, small and frequent releases provide upgraded solutions to the users and keep them involved in the project. Frequent releases also facilitate other practices, such as immediate feedback and continual integration.

Forty-Hour Week and Coding Standards These final two practices set the tone for how the developers should work. The exact number of hours a developer works isn’t the issue. The issue is that the project shouldn’t be a death march that burns out every member of the team. Neither should the project be a haphazard coding exercise. Developers should follow standards for coding and documentation. XP uses just the engineering principles that are appropriate for an adaptive process based on empirical controls.

■ XP Project Activities

Figure 10-16 shows an overview of the XP system development approach. It is divided into three levels: system (the outer ring), release (the middle ring), and iteration (the inner ring). System-level activities occur once during each development

FIGURE 10-16 XP development approach



project. A system is delivered to users in multiple stages called releases. Each release is a fully functional system that performs a subset of the full system requirements. A release is developed and tested within a period of no more than a few weeks or months. The activities in the middle ring cycle multiple times—once for each release. Releases are divided into multiple iterations. During each iteration, developers code and test a specific functional subset of a release. Iterations are coded and tested in a few days or weeks. There are multiple iterations within each release, so the iteration ring (inner) cycles multiple times.

The first XP development activity is creating user stories, which are similar to use cases in object-oriented analysis. A team of developers and users quickly documents all the user stories the system will support. Developers then create a class diagram to represent objects of interest within the user stories.

Developers and users then create a set of acceptance tests for each user story. Releases that pass the acceptance tests are considered finished. The final system-level activity is to create a development plan for a series of releases. The first release supports a subset of the user stories, and subsequent releases add support for additional stories. Each release is delivered to users and performs real work, thus providing an additional level of testing and feedback.

The first release-level activity is planning a series of iterations. Each iteration focuses on a small (possibly just one) system function or user story. The iterations' small size allows developers to code and test them within a few days. A typical release is developed by using from a few to a few dozen iterations.

After the iteration plan is complete, work begins on the first iteration-level activity. Code units are divided among multiple programming teams, and each team develops and tests its own code. XP recommends a test-first approach to coding. Test code is written before system code. As code modules pass unit testing, they are combined into larger units for integration testing. When an iteration passes integration testing, work begins on the next iteration.

When all iterations of a release have been completed, the release undergoes acceptance testing. If a release fails acceptance testing, the team returns it to the iteration level for repair. Releases that pass acceptance testing are delivered to end users, and work begins on the next release. When acceptance testing of the final release is completed, the development project is finished.

■ Scrum

Those of you who are familiar with rugby are aware that when a team gets possession of the ball, it attempts to go the entire distance in one continuous play—from point of possession to the score. The team works together, passing the ball back and forth; even when tackled, it can maintain possession and keep the ball in play. Originally, this “rugby” approach was applied to product development.

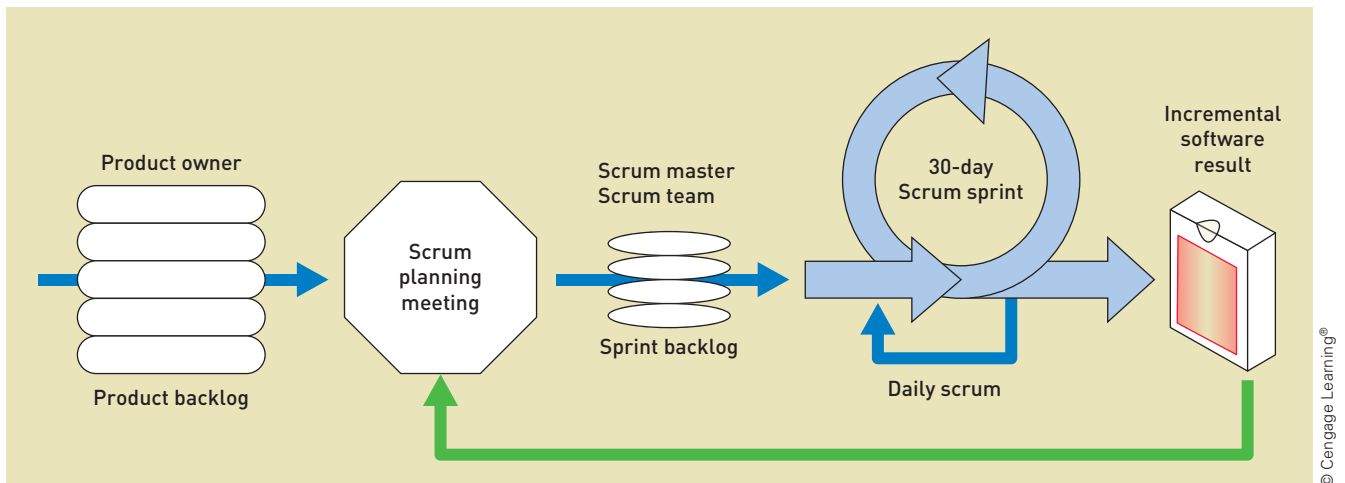
One interesting element in rugby is a *scrum*, which is used to get a ball back into play after a penalty. The defining characteristics of a scrum are that it begins quickly, is a very intense effort, involves the entire team, and usually only lasts for a short duration.

Combining some of these principles of rugby with the Agile philosophy gave rise to a methodology—the objective of which is to be quick, agile, and intense and to go the entire distance. This methodology is referred to as the Scrum approach. Over time, the techniques have been refined to fit into a powerful adaptive software development methodology. **Figure 10-17** illustrates an overview of the Scrum approach. There are three important Scrum areas to understand: the philosophy, the organization, and the practices.

■ Scrum Philosophy

The Scrum philosophy is also based on the Agile development principles described earlier. Scrum is responsive to a highly changing, dynamic environment

FIGURE 10-17 Scrum software development process



in which users might not know exactly what is needed and might also change priorities frequently. In this type of environment, changes are so numerous that projects can bog down and never reach completion. Scrum excels in this type of situation.

Scrum focuses primarily on the team level. It is a type of social engineering that emphasizes individuals more than processes and describes how teams of developers can work together to build software in a series of short mini-projects. Key to this philosophy is the complete control a team exerts over its own organization and its work processes. Software is developed incrementally, and controls are imposed empirically—by focusing on things that can be accomplished.

The basic control mechanism for a Scrum project is a list of all the things the system should include and address. This list—called the **product backlog**—includes user functions (such as use cases), features (such as security), and technology (such as platforms). The product backlog list is continually being prioritized, and only a few of the high-priority items are worked on at a time, according to the current needs of the project and its sponsor.

■ Scrum Organization

The three main organizational elements that affect a Scrum project are the **product owner**, the **Scrum master**, and the Scrum team or teams.

The product owner is the client, but he or she has additional responsibilities. Remember that in Agile development, the user and client are closely involved in the project. In Scrum, the product owner maintains the product backlog list. For any function to be included in the final system, it must first be placed on the product backlog. Because the product owner maintains that list, any request must first be approved and agreed to by the product owner. In traditional development projects, the project team initiates the interviews and other activities to identify and define requirements. In a Scrum project, the primary client controls the requirements. This forces the client and user to be intimately involved in the project. Nothing can be accomplished until the product owner creates the backlog.

The Scrum master enforces Scrum practices and helps the team complete its work. A Scrum master is comparable to a project manager in other approaches. However, because the team is self-organizing and no overall project schedule exists, the Scrum master's duties are slightly different. He or she is the focal point for communication and progress reporting—just as in a traditional

product backlog a prioritized list of user requirements used to choose work to be done in a Scrum project

product owner the client stakeholder for whom the system is being built

Scrum master the person in charge of a Scrum project—similar to a project manager

project. But the Scrum master doesn't set the schedule or assign tasks. The team does. One of the primary duties of the Scrum master is to remove impediments so the team can do its work. In other words, the Scrum master is a facilitator.

The Scrum team is a small group of developers—typically five to nine people—who work together to produce the software. For projects that are very large, the work should be partitioned and delegated to smaller teams. If necessary, the Scrum masters from all the teams can coordinate multiple team activities.

The Scrum team sets its own goal for what it can accomplish in a specific period of time. It then organizes itself and parcels out the work to members. In a small team, it is much easier to sit around a table, decide what needs to be done, and have members of the team volunteer or accept pieces of work.

■ Scrum Practices

The Scrum practices are the mechanics of how a project progresses. Of course, the practices are based on the Scrum philosophy and organization. The basic work process is called a **sprint**, and all other practices are focused on supporting a sprint.

sprint a time-controlled mini-project that implements a specific portion of a system

A Scrum sprint is a firm period called a *time box*, with a specific goal or deliverable. At the beginning of a sprint, the team gathers for a one-day planning session. In this session, the team decides on the major goal for the sprint. The goal draws from several items on the prioritized product backlog list. The team decides how many of the highest-priority items it can accomplish within the sprint. Sometimes, lower-priority items can be included for very little additional effort and can be added to the deliverables for the sprint.

After the team has agreed on a goal and has selected items from the backlog list, it begins work. The scope of that sprint is then frozen, and no one can change it—neither the product owner nor any other users. If users do find new functions they want to add, they put them on the product backlog list for the next sprint. If team members determine that they can't accomplish everything in their goal, they can reduce the scope for that sprint. However, the time period is kept constant.

Every day during the sprint, the Scrum master holds a daily Scrum, which is a meeting of all members of the team. The objective is to report progress. The meeting is limited to 15 minutes or some other short time period. Members of the team answer only three questions:

- What have you done since the last daily Scrum (during the last 24 hours)?
- What will you do by the next daily Scrum?
- What kept you or is keeping you from completing your work?

The purpose of this meeting is simply to report issues, not to solve them. Individual team members collaborate and resolve problems after the meeting as part of the normal workday. One of the major responsibilities of the Scrum master is to note the impediments and see that they are removed. A good Scrum master clears impediments rapidly. The Scrum master also protects the team from any intrusions. The team members are then free to accomplish their work. Team members do talk with users to obtain requirements, and users are involved in the sprint's work. However, users can't change the items being worked on from the backlog list or change the intended scope of any item without putting it on the backlog list.

At the end of each sprint, the agreed-on deliverable is produced. A final half-day review meeting is scheduled to recap progress and identify changes that need to be made for the following sprints. By time-boxing these activities—the planning, the sprint, the daily Scrum, and the Scrum review—the process becomes a well-defined template to which the team easily conforms, which contributes to the success of Scrum projects.

CHAPTER SUMMARY

System development projects are organized around a system development life cycle (SDLC). Some SDLCs are based on a more predictive approach to the project, and other SDLCs are based on a more adaptive approach. The predictive approach to the SDLC includes phases that are completed sequentially or with some overlap. The traditional SDLC phases are project initialization, project planning, analysis, design, implementation, deployment, and support. The adaptive approach to the SDLC is used when the requirements or technology are less certain and it is difficult to plan everything about the project in advance. Adaptive SDLCs use multiple iterations that allow the analysis, design, and implementation of smaller parts of the application to be completed and evaluated. The SDLC used in this text is an example of an adaptive SDLC, and the six core processes correspond to the phases of the traditional predictive SDLC. All development projects use an SDLC to manage the project, but there is more to system development than the SDLC. Models, techniques, and tools make up a system development methodology that provides guidelines for completing every activity in the SDLC.

Agile development, the leading trend in system development, helps keep system development projects responsive to change. It is a philosophy that values

change over following a plan, individuals over process and tools, working software over documentation, and customer collaboration over contract negotiation. Agile modeling describes principles for keeping a project agile.

The most formal adaptive, Agile development methodology is the Unified Process (UP). It was one of the first methodologies to be formalized with specific definitions for iterations and processes. Other more radical adaptive, agile methodologies are now being promoted and used. Two of the more popular ones are Extreme Programming and Scrum.

Extreme Programming (XP) and Scrum are methodologies that embody the most Agile principles. Two core elements of XP are that system tests are written first and that programmers work in pairs to design, code, and test the software. Thus, when a function is completed, it has not only been designed and coded, but it has also been reviewed and tested.

The Scrum approach defines a specific goal that can be completed within a short sprint. During that sprint, the project team is protected from all outside distractions so it can complete the defined goal. A product backlog of all outstanding requests is maintained by the client, and changes to the work the team is doing are only allowed between sprints.

KEY TERMS

adaptive approach to the SDLC
Agile development
Agile modeling (AM)
chaordic
incremental development
integrated development environments (IDEs)

pair programming
phases
predictive approach to the SDLC
product backlog
product owner
refactoring
Scrum master

sprint
technique
tool
UP discipline
visual modeling tools
walking skeleton
waterfall model

REVIEW QUESTIONS

1. What is a project?
2. What is the range of sizes of an information system development project?
3. What is the system development life cycle (SDLC)?
4. What characteristics of a project call for a predictive approach to the SDLC?
5. What characteristics of a project call for an adaptive approach to the SDLC?
6. What are the six phases of the traditional predictive SDLC?
7. Explain how the waterfall model of the SDLC controls the changes that occur during a project.
8. Explain the advantages of having the phases of a predictive SDLC overlap.
9. What organizing concept is included in all adaptive SDLCs?

10. For an adaptive SDLC, explain what goes on during each iteration.
11. The SDLC used in this text is based on what adaptive SDLC?
12. What are the core processes in the SDLC used in this book, and what traditional predictive SDLC phase corresponds to each process?
13. What is the iterative approach that involves completing and deploying part of an application over a few iterations and then completing and deploying another part of that application after a few more iterations?
14. Why do adaptive SDLCs not explicitly include the support phase?
15. What is a system development methodology?
16. What are some examples of models included in a methodology?
17. What are some examples of techniques included in a methodology?
18. What are some examples of tools included in a methodology?
19. What is Agile development?
20. What are the four “values” reflected in Agile development?
21. What is Agile modeling (AM)?
22. What are the 12 Agile modeling principles?
23. What are the four UP phases, and what is the objective of each?
24. What are the six UP development disciplines?
25. What are the three UP support disciplines?
26. Why is the word *extreme* included as part of Extreme Programming?
27. List the core values of XP.
28. List the XP practices.
29. What is the product backlog used for in a Scrum project?
30. Explain how a Scrum sprint works.

PROBLEMS AND EXERCISES

1. Write a one-page paper that distinguishes among the fundamental purposes of the analysis phase, the design phase, and the implementation phase of the traditional predictive SDLC.
2. Describe an information system project that might have three subsystems. Discuss how three iterations might be used for the project.
3. Why might it make sense to teach analysis and design phases and activities sequentially, like a waterfall, even though iterations are, in practice, used in nearly all development projects?
4. List some of the models that architects create to show different aspects of a house they are designing. Explain why several models are needed.
5. What models might an automotive designer use to show different aspects of a car?
6. Sketch and write a description of the layout of your room at home. Are both the sketch and the written description considered models of your room? Which is more accurate? More detailed? Which would be easier to follow by someone unfamiliar with your room?
7. Describe a technique you use to help you complete the activity “Get to class on time.” What are some of the tools you use with this technique?
8. Describe a technique you use to make sure you get assignments done on time. What are some of the tools you use with this technique?
9. What are some other techniques you use to help you complete activities in your life?
10. Go to the campus placement office to gather some information on companies that recruit information systems graduates. Try to find any information about the companies’ approaches to developing systems. Is their SDLC described? Do any mention an IDE or a visual modeling tool? Visit the companies’ Web sites to look for more information.
11. Visit the Web sites of a few leading information systems consulting firms. Try to find information about their approaches to developing systems. Are their SDLCs described? Do the sites mention any tools, models, or techniques?
12. The Unified Process (UP) was first developed by a company called Rational, which is now owned by IBM. On the IBM Web site, find any information about UP tools available through IBM/Rational. Briefly describe the suite of tools available. Also, look on the IBM Web site and other Web sites (such as the Agile Modeling Web site) for opinions on the relationships and commonality between the UP and Agile modeling. Report your findings.
13. Consider XP’s team-based programming approach in general and its principle of allowing any programmer to modify any code at any time in particular. No other development approach or programming management technique follows

this particular principle. Why not? In other words, what are the possible negative implications of this principle? How does XP minimize these negative implications?

14. Visit the Web sites of the Agile Alliance (www.agilealliance.org) and Agile Modeling (www.agilemodeling.com). Find some articles on project management in an Agile environment. Summarize key points that you think make project management more difficult in this environment than in a traditional, predictive project. Do the same for key points that make project management easier for an Agile project.
15. Find someone in your community who is working on a software development project that is using Agile principles. How was the team trained to use Agile development? How was this approach adopted in the organization? What is the general feeling about its success? What aspects does this developer like? Which aspects does he or she find frustrating or difficult to use?
16. Research on the Internet one of the other techniques such as *Lean Development* or *Feature Driven Development*. Write a short summary of its principles and practices. Comment on its strengths and weaknesses.

CASE STUDY

A “College Education Completion” Methodology

Given that you are reading this book, you are probably a college student working on a degree. Think about completing college as a project—a big project lasting many years and costing more than you might want to admit. Some students do a better job managing their college completion projects than others. Many fail entirely (certainly not you), and most complete college late and way over budget (again, certainly not you).

As with any other project, to be successful, you need to follow some sort of “college education completion” methodology—that is, a comprehensive set of guidelines

for completing activities and tasks from the beginning of planning for college through to the successful completion.

1. What are the phases that your college education completion life cycle might have?
2. What are some of the activities included with each phase?
3. What are some of the techniques you might use to help complete those activities?
4. What models might you create? Differentiate the models you create to get you through college from those that help you plan and control the process of completing college.
5. What are some of the tools you might use to help you complete the models?

RUNNING CASE STUDIES

Community Board of Realtors®

The Board of Realtors Multiple Listing Service (MLS) system isn’t very large in terms of use cases and domain classes. In that respect, the functional requirements are simple and well understood. MLS needs a Web site with public access to the listings, and it also needs to allow agents and brokers to log in to the system to add and update listings. There is very little back-end administrative data maintenance required, except to add or update a real estate office or agent.

1. Compared to the Tradeshow application described in Chapter 1, how long might this project take, and which approach to the SDLC would be most appropriate?
2. If you use a predictive SDLC, how much time might each phase of the project take? How much overlap of phases might you plan for? Be specific about how you would overlap the phases.

3. If you use an adaptive SDLC, how many iterations might you plan to include? What use cases would you analyze, design, and implement in the first iteration? What use cases would you work on in the second iteration? In additional iterations? Think in terms of getting the core functionality implemented early and then building the supporting functionality.
4. Assume this project focused on Web access to the MLS. If you also plan to deploy a smartphone application for use by the public and by the agents and brokers, how might this affect your choice of the approach to the SDLC?

The Spring Breaks 'R' Us Travel Service

Recall from Chapter 2 that SBRU's initial system included four major subsystems: Resort relations, Student booking, Accounting and finance, and Social networking. The project calls for an adaptive approach to the SDLC for several reasons. One, it is relatively large in scope. Two, there is a diverse set of users in several functional areas, internal and external to the company and in several foreign countries. Three, the project needs to use an assortment of newer technologies that can communicate anytime and anywhere.

1. The SBRU information system includes four major subsystems: Resort relations, Student booking, Accounting and finance, and Social networking. Although you have only worked with the domain model class diagram for the Social networking subsystem, list as many of the domain classes that would probably be involved

On the Spot Courier Services

In the On the Spot system, package pickup and delivery are closely integrated with route schedules. However, recall the RMO system, where there is a Sales subsystem, an Order fulfillment subsystem, a Customer account subsystem, and a Marketing subsystem. You could conceive of the On the Spot system as also consisting of four subsystems:

- Customer account subsystem (like customer account)
- Pickup request subsystem (like sales)
- Package delivery subsystem (like order fulfillment)
- Routing and scheduling subsystem

Assuming that On the Spot's system developer approached this new system from this point of view and that the developer also decided to use an adaptive, iterative approach, answer these questions:

1. In what order would you develop the four subsystems? Support your answer.

What are the implications for including the smartphone application in the initial project versus having a separate project for wireless later?

5. Consider using incremental development to include the Web application and the wireless support. Describe what would be included in the first and second deployments of the project. Take into consideration that you might want to work on some initial problem solving for requirements, design, and implementation of the wireless support at the same time you are working on the Web application.

in each of the subsystems. Note which classes are used by more than one subsystem.

2. Based on the overlapping classes, what domain classes seem to be part of the core functionality for SBRU? Draw a domain model class diagram that shows these classes and their associations.
3. Suppose you plan to implement the basic use cases that create and maintain the classes that are part of the core functionality you just modeled. Describe what domain classes you would focus on in each iteration if you assumed that you would need two iterations for the initial core functionality and two additional iterations to complete each of the subsystems.
4. How might you use incremental development to get some core functionality or some subsystems deployed and put into use before the project is completed?

2. Reviewing your work from Chapter 3, assign each of your use cases to a particular subsystem. Does this change your answer or does it strengthen your original premise? Support your answer.
3. Reviewing your work from Chapter 4, assign each of your classes to a subsystem. (Note: Some classes may be in multiple subsystems. The primary subsystem is the one that "creates" the objects in that class.) Does this change your answer or does it strengthen your original premise? Support your answer.
4. Considering the Agile modeling principles, discuss each of the following:
 - a. In Chapter 3, you developed a list of use cases and a use case diagram. If you follow the Agile modeling philosophy, how much or how little of this model do you think is necessary? Support your answer.

- b. In Chapter 4, you developed a class diagram. If you follow the Agile modeling philosophy, how much or how little of this model do you think is necessary? Support your answer.
- c. In Chapter 5, you developed some use case descriptions, activity diagrams, and system sequence diagrams. If you follow the Agile modeling philosophy, how many or how few

of these models do you think are necessary? Support your answer.

- d. In Chapter 6, you learned about system design and system environments. If you follow the Agile modeling philosophy, how many or how few of these models do you think are necessary? Support your answer.

Sandia Medical Devices Real-Time Glucose Monitoring

Review the original system description in previous chapters and the use case diagram shown in **Figure 10-18** to familiarize yourself with the proposed system. Consider this additional information:

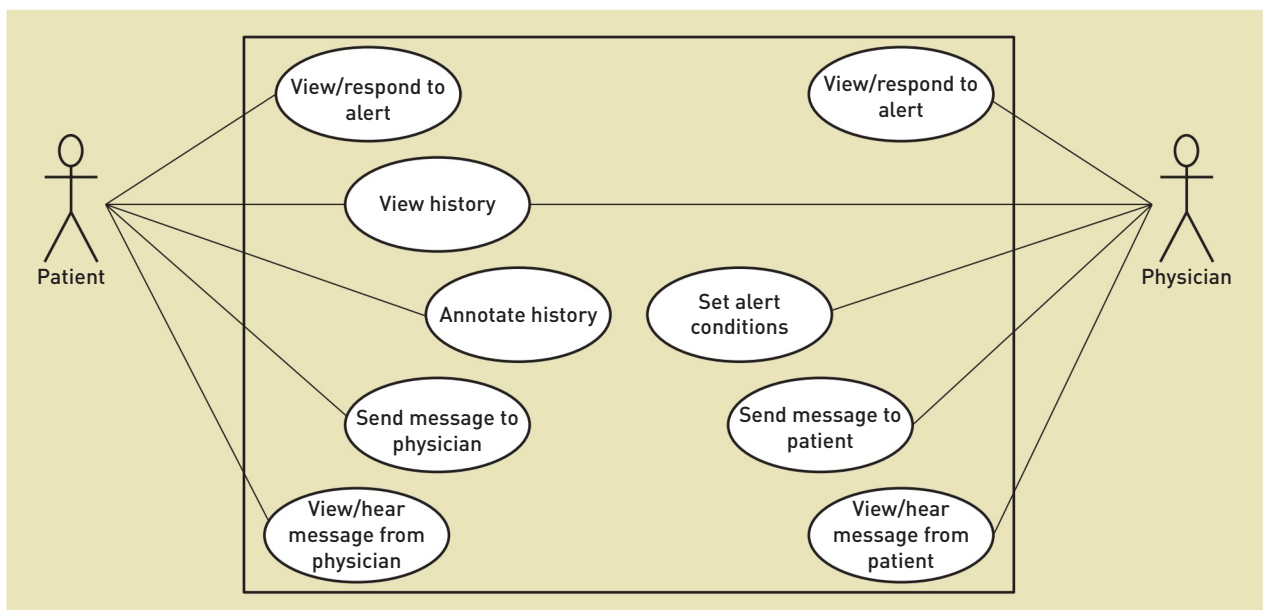
- Sandia Medical Devices (SMD) and New Mexico Health Systems (NMHS) are developing the system jointly. Project staff will include analysts, designers, and programmers from both organizations. Three technical staff members from each organization have been assigned initially, and the budget includes sufficient funds to add other personnel for short-term assignments as needed. In addition, NMHS will assign a physician and a physician's assistant to the project one day per week.
- It is anticipated that SMD personnel assigned to the project will work primarily at NMHS facilities in office space and with computer equipment dedicated to developing the Real-Time Glucose Monitoring (RTGM) system.
- NMHS anticipates recruiting a handful of its own diabetic employees to provide requirements and to test the prototype RTGM software.

- SMD and NMHS anticipate a six-month development schedule for an initial version of the server software and Android-based client-side software. That will be followed by a three-month period for evaluation and another three-month period for development of improved software versions and support for a wider range of mobile phone operating systems.

Answer these questions:

1. Given the system goals, requirements, and scope as they are currently understood, is the project schedule reasonable? Why or why not?
2. How well understood are the system requirements at the start of the project? What are the implications of your answer for using a predictive, adaptive, or mixed SDLC? What are the implications of your answer for using Agile techniques?
3. Medical personnel at NMHS have very busy schedules. NMHS's decision to assign two medical practitioners to the project for one day a week represents a significant investment in salary and lost revenue. How should project iterations be structured to ensure rapid progress to completion, high quality, and efficient use of medical practitioner time?

FIGURE 10-18 Use cases for the patient and physician actors



FURTHER RESOURCES

- Agile Alliance, www.agilealliance.org.
- Scott W. Ambler, *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. John Wiley & Sons, 2002.
- Scott Ambler, John Nalbone, and Michael J. Vizdos, *The Enterprise Unified Process: Extending the Rational Unified Process*. Prentice Hall, 2005.
- Ken Auer and Roy Miller, *Extreme Programming Applied: Playing to Win*. Addison-Wesley, 2002.
- D. E. Avison and G. Fitzgerald, *Information Systems Development: Methodologies, Techniques, and Tools* (3rd ed.). McGraw-Hill, 2003.
- Kent Beck, *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.
- Mike Cohn, *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley, 2010.
- Philippe Kruchten, *The Rational Unified Process: An Introduction*. Addison-Wesley, 2004.
- Craig Larman, *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley, 2004.
- "Manifesto for Agile Software Development," Agile Alliance, www.agilemanifesto.org.
- Pete McBreen, *Questioning Extreme Programming*. Addison-Wesley, 2003.
- Andrew Pham and Phuong-Van Pham, *Scrum in Action*. Course Technology, 2011.
- Ken Schwaber and Mike Beedle, *Agile Software Development with Scrum*. Prentice Hall, 2002.

CHAPTER ELEVEN

CHAPTER OUTLINE

- ▶ Principles of Project Management
- ▶ Activities of Core Process 1: Identify the Problem and Obtain Approval
- ▶ Activities of Core Process 2: Plan and Monitor the Project

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- ▶ Describe the factors that cause a software development project to succeed or fail
- ▶ Describe the responsibilities of a project manager
- ▶ Describe the knowledge areas in the project management body of knowledge (PMBOK)
- ▶ Describe the Agile approach to the project management knowledge areas
- ▶ Explain the activities required to get a project approved (Core Process 1)
- ▶ Explain the activities required to plan and monitor a project (Core Process 2)

OPENING CASE BLUE SKY MUTUAL FUNDS: A NEW DEVELOPMENT APPROACH

Jim Williams, vice president of finance for Blue Sky Mutual Funds, spoke first. “There are some things I like about this new approach, but other things worry me,” he told Gary Johnson, the company’s director of information technology.

“This idea of ‘growing’ the system through several iterations makes a lot of sense to me. It is always hard for my people to know exactly what they need a new information system to do and what will work best for the company. So, if they can get their hands on the system early, they can begin acceptance testing and try it out to see whether it addresses their needs in the best way.”

“Let me see if I understand the big picture, though. Your development team and my investment advisors will decide on a few core processes that the system needs to support and then your team will design and build a system to support those core processes. You will do that in a mini-project that will last about six weeks. Then, you will continue adding more functionality through several other mini-projects until the system is complete and functioning well. Is that right?”

Jim was becoming more enthusiastic about this new approach to system development.

“Yes, that’s the basic idea,” Gary said. “Your users need to understand that the first few versions of the system won’t be complete and may not be completely robust either. But these early versions will give them something to work with and try out. We also need good feedback from their acceptance testing so the system will be thoroughly tested by the time we are through.”

“I realize that,” Jim said. “My people will like not having to think from the very beginning about everything they need the system to do. They’ll like being able to try things out. As I said earlier, I like this approach. However, the part I don’t like about this approach is that it will be more difficult for you to give me a firm time schedule and project cost. That worries me. In the past, those have been two of the major tools we used to monitor a project’s progress. Are you saying that now we won’t have a schedule at all? And you want an open budget?” Jim frowned.

“It’s not as bad as it first sounds,” Gary said. “This approach is an ‘adaptive’ approach, by which I mean that because the system is growing, the project is more open ended. The project manager will still create a schedule and estimate the project costs, but she won’t even try to identify and lock in all the required functionality for several of the iterations. Because the system’s scope is going to continually be refined over the first few iterations, there is the risk of ‘scope creep.’ That is one of the biggest risks with adaptive approaches. You and I should meet with the project manager fairly frequently to ensure that the scope is controlled and the project doesn’t get out of control.”

“Okay,” Jim said. “You have convinced me to try this new approach. However, let’s treat this project as a pilot and see how it works. If it’s successful, we will consider using this iterative approach on our other projects.” Jim and Gary agreed that a pilot was the best way to get started. Gary then headed off to meet with the project manager and get the project started.

■ Overview

Chapter 10 introduced you to the SDLC and the various alternatives for organizing software development activities. By now, you may be asking yourself such questions as:

- How are all these activities coordinated?
- How do I know which tasks to do first?
- How is the work assigned to the different teams and team members?
- How do I know which parts of the new system should be developed first?

The purpose of project planning and project management is to bring some order to all these (sometimes seemingly unrelated) tasks. As you will learn in this chapter, the success of any given project highly depends on the skills and abilities of those managing the project. You will also learn that project management skills aren’t only for project managers—that all the project team members contribute to the management of the project and thus to its success.

This chapter first discusses the need for project management and the principles associated it. The rest of this chapter discusses the detailed activities that are associated with the first two core processes of system development, both of

which are primarily project management processes. The purpose of this chapter is to teach you how to plan, organize, and direct a system development project.

■ Principles of Project Management

Many of you may have built a Web page with HTML or written a computer program for yourself or a friend. In those cases, where it was just you working, you weren't too concerned about how to organize your work or how to manage the project. However, as soon as two or more developers are working together, the work must be partitioned and organized, with specific assignments for each developer. This is true whether the project uses a predictive approach or an adaptive approach. As discussed in the last chapter, the chosen methodology lays out a complex set of activities and tasks that must be carefully managed. Failing to organize usually causes wasted time and effort as well as confusion and may even cause the project to fail.

Even though every project team designates one person as the project manager, with primary responsibility for the way the team functions, all members contribute to the team's management. The project manager for the RMO Consolidated Sales and Marketing System (CSMS) project is Barbara Halifax, but she has a senior systems analyst helping her every step of the way. As the project proceeds, all team members are involved in aspects of managing the project.

As discussed in earlier chapters, a project is a planned undertaking with a beginning and an end, which produces a predetermined result and is usually constrained by a schedule and resources. The development of information systems fits this definition. In addition, it is usually a quite complex project, with many people and tasks that have to be organized and coordinated. Whatever its objective, each project is unique. Different products are produced, different activities are required with varying schedules, and different resources are used. This uniqueness makes information systems projects difficult to control.

■ The Need for Project Management

History is replete with stories of software development projects that go awry. Probably one of the most visible examples of a less-than-successful development project was the Affordable Care Act enrollment system—frequently referred to as Obamacare. Even though it was reported on the news media that the only problem was that it could not handle the volume of traffic, more in-depth evaluation uncovered several serious flaws. According to Ben Simo, the past president of the Association for Software Testing, there were other serious problems. One small example:

“In mid-October, he went to Healthcare.gov to help a family member get insurance, only to find his progress blocked. When he investigated the cause, he discovered that one part of the website had created so much ‘cookie’ tracking data that it appeared to exceed the site’s capacity to accept his login information. That’s the mark of a fractured development team.” (<http://swampland.time.com/2013/10/24/traffic-didnt-crash-the-obamacare-site-alone-bad-coding-did-too/>)

A “fractured development team” really means that the project manager and team leaders were unable to manage the project. Admittedly, it was a giant project under a very tight deadline with many diverse stakeholders—as are many software projects.

There are a number of organizations that study software development to evaluate success rates and to identify best practices. Software development success is often measured using three criteria: finishing on time, finishing within budget,

and effectively meeting the need as expressed by the original problem definition. Given these three criteria, software projects are often categorized in three ways:

1. Successful projects, which are completed on time and within budget while meeting the users' requirements for functionality
2. Challenged projects, which have some combination of being late, overbudget, or reduction of scope
3. Failed projects, which are canceled or result in the system never being used

One of the early organizations to study software development success was the Standish Group, which started surveying success rates back in 1995. At that time, the results were dismal. One-third of all projects ended up as failed projects that were canceled. Fully one-half of projects were challenged and were late, overbudget, and often with reduced functionality. Only about 15 percent were considered successful.

Over the last 20 years, there has been a major effort to improve these numbers. You learned in the previous chapter about new development paradigms that are easier to manage and respond better to users' needs. **Figure 11-1** illustrates success rates based on several different types of development paradigms. Even though the success has improved substantially, billions of dollars are still spent on projects that don't meet their objectives.

Many of these studies and reports don't just indicate the rate of information technology (IT) project failure or success. They also identify the reasons for each. Here are some of the primary reasons for failure:

- Undefined project management practices
- Poor IT management and poor IT procedures
- Inadequate executive support for the project
- Inexperienced project managers
- Unclear business needs and project objectives
- Inadequate user involvement

It is notable that the primary reasons projects fail are a lack of executive involvement and a lack of management skills. The other major reason is lack of involvement by the user community. In other words, projects don't tend to fail for lack of programming skills or enthusiastic developers.

For an IT project to be successful, strong IT management and business direction need to be present. The other major element in all project success is sound project management procedures as well as experienced and competent project managers. In fact, good project managers always ensure that they have received clear directives from business executives and committed user involvement with the requirements for the new system. Substantial research and improvement have also occurred in project management approaches. You will learn about specific project management techniques that improve project success.

FIGURE 11-1 Project success rates for various development paradigms

Development paradigm	Successful	Challenged	Failed
Ad hoc	50%	35%	15%
Traditional	49%	32%	18%
Lean	72%	21%	7%
Agile	64%	30%	6%
Iterative	65%	28%	7%

© Cengage Learning®

Source: <http://www.drdoobs.com/architecture-and-design/the-non-existent-software-crisis-debunki/240165910>.

project management organizing and directing other people to achieve a planned result within a predetermined schedule and budget

■ The Role of the Project Manager

Project management is organizing and directing other people to achieve a planned result within a predetermined schedule and budget. At the beginning of a project, a plan is developed that specifies the activities that must take place, the deliverables that must be produced, and the resources that are needed. Even though the project plan is quite different depending on whether a predictive traditional approach is used or whether an adaptive approach is used, successful projects always have some type of project plan. Project management can also be defined as the processes or activities used to plan the project and then to monitor and control it.

One of the most exciting careers for IT-oriented people is being a project manager. As projects become more complex because of shorter time frames, distributed project teams (including offshore and cross-cultural teams), rapidly changing technology, and more sophisticated requirements, highly qualified project managers are sought after and paid well. Many universities have added project management courses to their curricula to respond to the needs of the industry. There is a strong need and a high demand for people who are capable project managers. As your career progresses, you should develop your management skills. You may even want to become active in the Project Management Institute (PMI), which is the most well-known professional organization for project managers.

Overall, project managers must be effective internally (managing people and resources) and externally (conducting public relations). Internally, the project manager serves as locus of control for the project team and all its activities. He or she establishes the team's structure so work can be accomplished. This list identifies a few of these internal responsibilities:

- Developing the project schedule
- Recruiting and training team members
- Assigning work to teams and team members
- Assessing project risks
- Monitoring and controlling project deliverables and milestones

Externally, the project manager is the main contact for the project. He or she must represent the team to the outside world and communicate the team members' needs. Major external responsibilities include:

- Reporting the project's status and progress
- Working directly with the client (the project's sponsor) and other stakeholders
- Identifying resource needs and obtaining resources

In Chapter 2, you learned about the various stakeholders of a system. These stakeholders were divided into internal stakeholders, which are those inside the organization, and external stakeholders, which are those outside of the organization. Drilling down to a more detailed level, you can identify those people who are part of the project team and work for the project manager, and those stakeholders who do not. Among the people who are not part of the project team are several groups of people that the project manager must interact with.

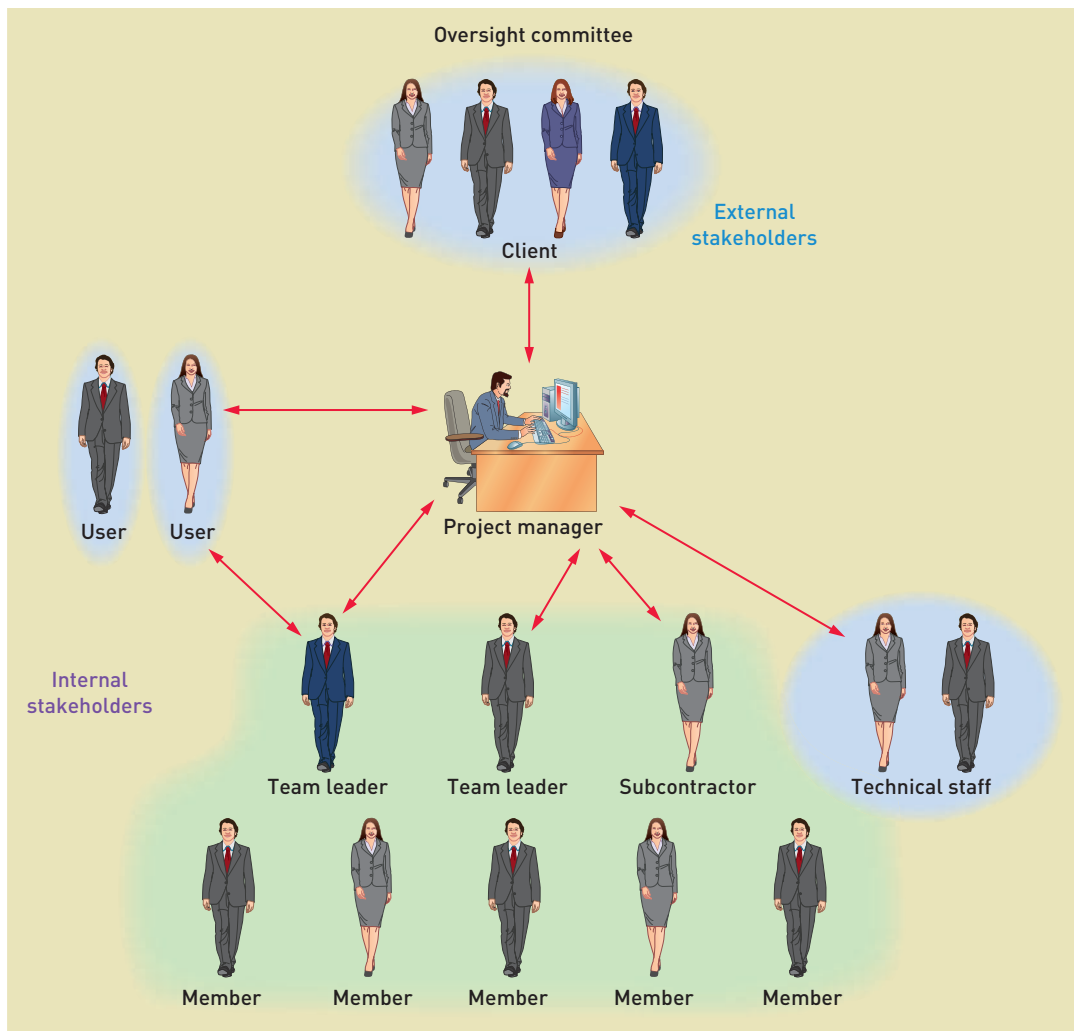
First, there is the **client** (i.e., the customer), who pays for the development of the new system. Project approval and the release of funds come from the client. For in-house developments, the client may be an executive committee or a vice president. The client approves and oversees the project, along with its funding. For large, mission-critical projects, an **oversight committee** (sometimes called the steering committee) may be formed. This consists of clients and other key executives who have a vision of the organization's strategic direction and a strong interest in the project's success. On the other hand, the **users** are the people who will actually use the new system. The user typically provides information about the detailed functions and operations needed in the new system.

client the person or group that funds the project

oversight committee clients and key managers who review the progress and direct the project

users the person or group of people who will use the new system

FIGURE 11-2 Stakeholders in a system development project



Communication with the client and oversight committee is an important part of the project manager's external responsibilities. Similarly, working with the team leaders, team members, internal technical staff, and any subcontractors is an important part of a project manager's internal responsibilities. The project manager must ensure that all internal and external communication is flowing properly. **Figure 11-2** depicts the various groups of people involved in a development project.

■ Project Management and Ceremony

Another dimension that has a heavy impact on project management is the level of formality, sometimes called ceremony, required for a given project. **Level of formality** or **ceremony** is a measure of the amount of documentation generated, the traceability of specifications, and the formality of the project's decision-making processes. Some projects, particularly small ones, are conducted with very low ceremony. Meetings occur in the hallway or around the water cooler. Written documentation, formal specifications, and detailed models are kept to a minimum. Developers and users usually work closely together on a daily basis to define requirements and develop the system. Other projects, usually larger, more complex ones, are executed with high ceremony. Meetings are often held on a predefined schedule, with specific participants, agendas, minutes, and follow-through. Specifications are formally documented with an

Level of formality or **ceremony** the rigor of holding formal meetings and producing detailed documentation

abundance of diagrams and documentation and are frequently verified through formal review meetings between developers and users.

A project's ceremony isn't the same as whether its approach is predictive or adaptive. However, even though the approach and ceremony are different, large predictive projects often tend to have high ceremony, with lots of meetings and documentation. Unfortunately, the extensive documentation tended to increase the length of the project and sometimes contributed to cost overruns. Techniques such as rapid application development (RAD) were utilized to help manage large predictive projects with less formality. This approach required less documentation and fewer status and review meetings. Of course, many smaller projects were often managed with less ceremony.

Adaptive projects can also be more or less formal in the way they are managed. The Unified Process, which was explained in Chapter 10, is quite formal, with high ceremony. Each iteration is precisely defined, with such specific outcomes as specifications, diagrams, prototypes, and deliverables. However, other adaptive approaches, such as lean, iterative, or Agile methods, lend themselves to being managed with much less formality. The inherent characteristics of an iterative approach, with its "just in time" project plans, easily adjust to less documentation, fewer diagrams for specifications, and less-formal status reporting.

■ Project Management Body of Knowledge (PMBOK)

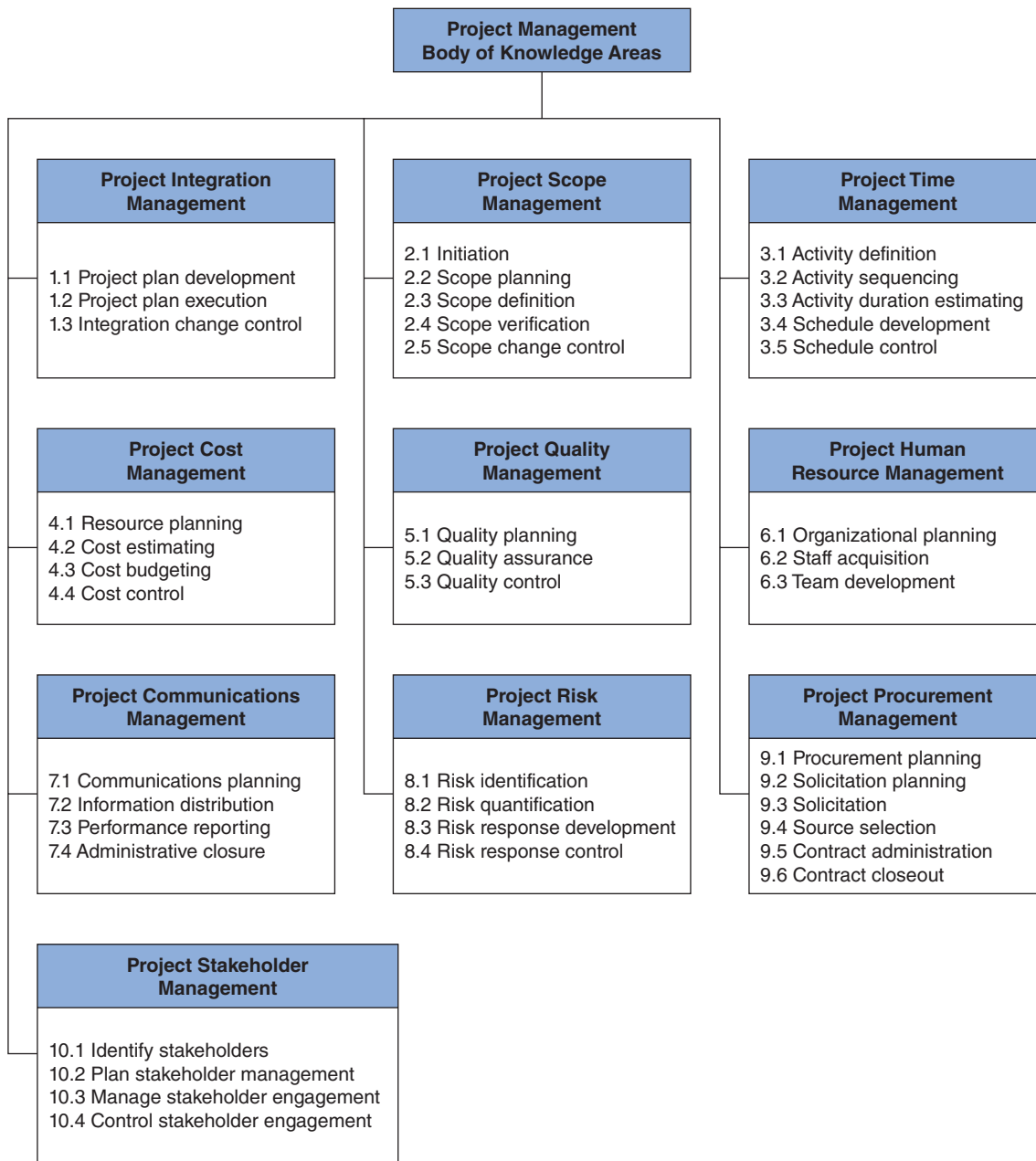
The PMI is a professional organization that promotes project management, primarily within the United States but also throughout the world. In addition, professional organizations in other countries promote project management. The PMI has a well-respected and rigorous certification program, and many corporations encourage their project managers to become certified.

As part of its mission, the PMI has defined a body of knowledge for project management. This body of knowledge, referred to as the **project management body of knowledge (PMBOK)**, is a widely accepted foundation of information that every project manager should know. The PMBOK is organized into 10 knowledge areas (**Figure 11-3**):

project management body of knowledge (PMBOK) a project management guide and standard of fundamental project management principles

- **Project Integration Management.** Integrating all the other knowledge areas into one seamless whole
- **Project Scope Management.** Defining and controlling the functions that are to be included in the system as well as the scope of the work to be done by the project team
- **Project Time Management.** Creating a detailed schedule of all project tasks and then monitoring the progress of the project against defined milestones
- **Project Cost Management.** Calculating the initial cost/benefit analysis and its later updates and monitoring expenditures as the project progresses
- **Project Quality Management.** Establishing a comprehensive plan for ensuring quality, which includes quality control activities for every phase of a project
- **Project Human Resource Management.** Recruiting and hiring project team members; training, motivating, and team building; and implementing related activities to ensure a happy, productive team
- **Project Communications Management.** Identifying all stakeholders and the key communications to each; also establishing all communications mechanisms and schedules
- **Project Risk Management.** Identifying and reviewing throughout the project all potential risks for failure and developing plans to reduce these risks
- **Project Procurement Management.** Developing requests for proposals, evaluating bids, writing contracts, and then monitoring vendor performance
- **Project Stakeholder Management.** Identifying and communicating with the stakeholders of the new system

FIGURE 11-3 Project management body of knowledge areas and processes



© Cengage Learning®

Reference: <http://www.pmi.org/PMBOK-Guide-and-Standards.aspx>

As you progress in your career, you would be wise to keep a record of the project management skills you observe in others as well as those you learn from your own experiences. One place to start is with the set of skills a systems analyst needs, as described in earlier chapters. A good project manager knows how to develop a plan, execute it, anticipate problems, and make adjustments. Project management skills *can* be learned. If project management is in your long-term game plan, you might consider joining PMI early in your career.

■ Agile Project Management (APM)

In the last chapter, you learned about the Agile approach to developing systems and the four values of Agile development, which tended to prefer flexibility over

plans and defined procedures. Obviously, these values have a large impact on the way a project is managed. However, one of the concerns with them is that they imply a working environment that has no controls or plans—one that can turn into pure chaos. Chapter 10 also introduced a term, *chaordic*, that describes a project that expects and allows chaos while remaining controlled or ordered.

One of the major challenges of Agile project management is how best to balance the flexibility and chaos of an Agile team with the order and control needed for a project. More than anything else, Agile project management is a way of balancing these two conflicting requirements: how to be agile and flexible while maintaining control of the project schedule, budget, and deliverables. In recent years, several implementations of Agile principles have appeared in the form of specific methodologies, each with their own set of principles, such as Scrum and Lean.

To help you understand Agile project management better, the following sections discuss five of the ten knowledge areas of the PMBOK and discuss the issues involved in implementing them by using Agile principles.

■ Agile Scope Management

Scope management refers to the scope of the new system and the scope of the project. In traditional predictive projects, the project manager and the team attempted to define the scope in both areas at the beginning of the project, during the planning phase. Unfortunately, for most new systems, there were so many unknowns that the scope was almost never defined accurately. The Agile philosophy accepts the fact that the scope isn't well understood and that there will be many changes, updates, and refinements to the requirements as the project progresses. However, uncontrolled scope can result in a project that never finishes, even if it is an Agile project. The project manager must have a process and mechanisms in place to control the scope of the project. How can this be done?

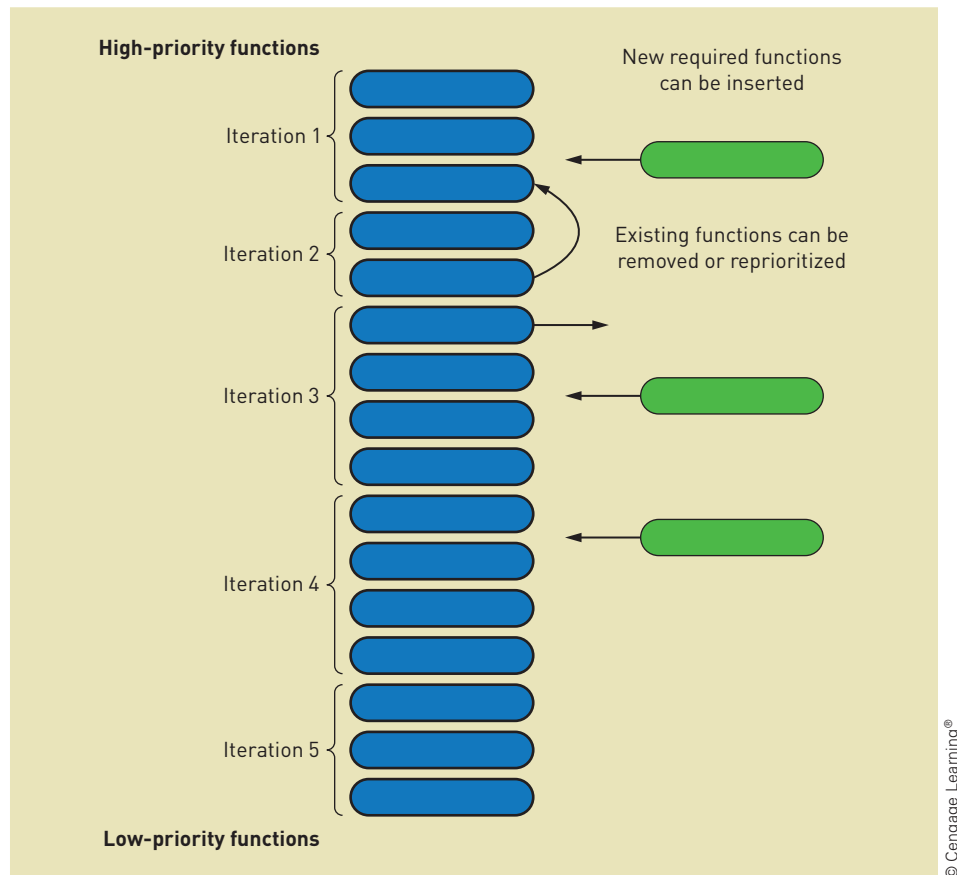
Let us assume that one of the major outcomes of the planning iteration was the decision to develop a prioritized list of business requirements that the new system needs to support. **Figure 11-4** represents this list, with the higher-priority items toward the top and the lower-priority items toward the bottom. These requirements can be prioritized by using several criteria, including importance to the business, risk, complexity, size, and other dependencies. In most projects, some combination of these criteria is used to prioritize the requirements. Figure 11-4 also indicates that the project team has made a preliminary assignment of these requirements to iterations. As new requirements are defined, they are prioritized, inserted into the stack, and assigned to an iteration.

Controlling the scope is a decision made by the client, with input provided by the project team and the users. With an iterative project, a deliverable is usually provided at the end of each iteration. Because the system is growing throughout the project, with the highest-priority requirements implemented first, the client is able to shut down the project when he or she feels that the system is complete enough to satisfy the business need. Most projects usually require one or two more iterations to do final integration and testing to ensure that the system will scale for high volume and that it meets all the “hardening” requirements for security purposes.

■ Agile Time Management

Traditional time management is primarily concerned with scheduling tasks: creating the schedule, assigning work according to the schedule, and monitoring progress against the schedule. In predictive projects, the schedule is created during the initial planning phase and entered into a project scheduling system, such as Microsoft Project.

FIGURE 11-4 *Scope management with changing requirements*



In an Agile project, because the requirements are always changing, it can be very difficult to create and maintain a meaningful project schedule. The initial planning effort will usually include the beginning set of requirements and divide the project into iterations, with a preliminary assignment of requirements to iterations. However, it is expected that the number of iterations and the assignments will change as new requirements are discovered and put on the prioritized stack.

Within an iteration, which often lasts from two to four weeks, a more detailed schedule can be developed. The Agile philosophy includes the idea that only for small work projects, in which the tasks are performed at nearly the same time (i.e., within one iteration), can a meaningful schedule be developed. In addition, the project team, not the project manager or team leader, will schedule its own work. Thus, for an Agile project, each iteration is usually planned as the first task within the iteration. The tasks are identified, estimates of the effort are developed, and work is assigned by the project team members. Because there are so many iterations in a project, the project team gets lots of practice and quickly becomes proficient at estimating and scheduling the work.

■ Agile Cost Management

It is normal for the client stakeholder to ask, “How long will it take and how much will it cost for this new system to be developed?” These questions are hard to answer. For predictive projects, the project manager gives estimates, but as you saw earlier, these are usually incorrect. Agile project managers admit more readily that time and cost estimates are difficult to make, especially with a project in which the requirements are expected to change throughout. Hence, estimating the project’s cost isn’t as important as controlling the

cost during the life of the project. The project manager’s responsibility to control costs is just as important for an Agile project as it is for a traditional predictive project.

■ Agile Risk Management

In most adaptive, iterative projects, including Agile projects, close attention is given to project risks, particularly technical risks. Iterative projects are often risk-driven, meaning that early iterations focus specifically on addressing the most critical project risks. Although a similar emphasis on risk can be included in a predictive project, it is more difficult to integrate specific risk-reducing activities into the project schedule. The major difference between the two types of projects is that in predictive projects, separate prototypes are built, whereas in adaptive projects, the high-risk portions of the new system are built first.

■ Agile Quality Management

Usually, quality management has to do with the quality of the deliverable from the project. However, in an Agile project, you also consider the quality of the process. How well is the project working, and how well do the internal procedures promote project success?

In a predictive project, the final set of tasks consists of the system test, the integration test, and the user acceptance test. However, scheduling these extensive tests at the end of the project renders it very difficult and expensive to make the necessary changes. An alternative is to deploy the system with minimal testing, which helps the budget but can cause many problems for the company.

In an Agile project, each iteration has a deliverable. Often, each iteration also integrates a new piece into the growing total system. Within each iteration, the new pieces are tested by themselves and as integrated with the rest of the system. The users also get involved in testing the system’s ability to meet their business needs. Hence, testing and quality control are spread across the entire project and usually provide a better-tested and more robust system.

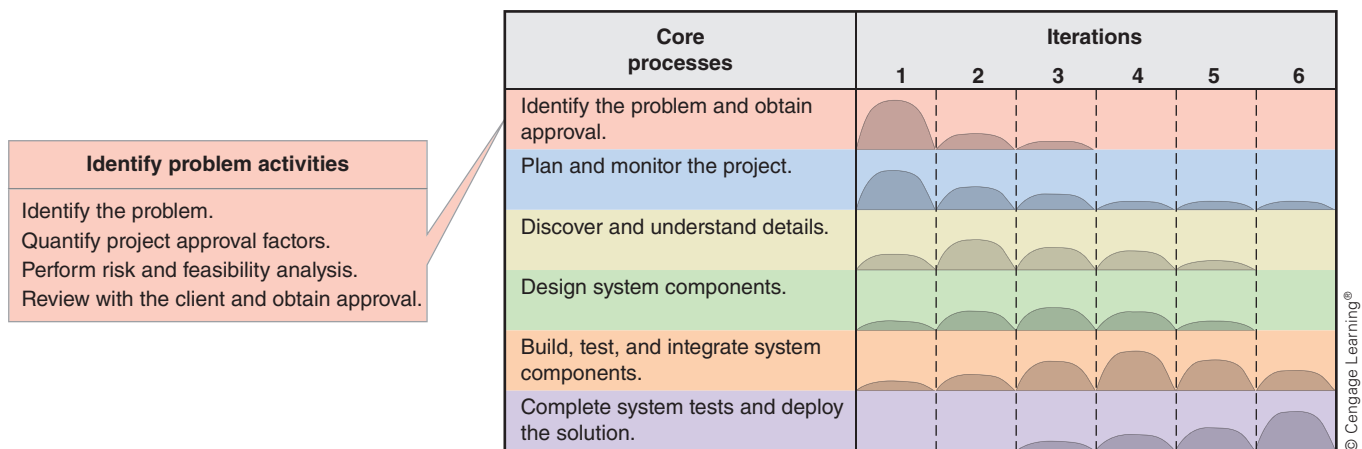
Another kind of quality control that should be done as part of an Agile project is a process evaluation at the end of each iteration. In other words, the project team does a self-evaluation to figure out how well it did and what could be done to improve the next iteration.

■ Activities of Core Process 1: Identify the Problem and Obtain Approval

Chapter 1 introduced you to the basic principles of iterative development using six core processes. In fact, the core processes are applicable to either a predictive approach or an adaptive approach. As explained in the last chapter, with a predictive approach, the core processes are performed in a waterfall sequence. In an iterative approach, these core processes are performed multiple times—during each iteration. Obviously, some overall project planning must be done at the initiation of the project, but detailed project planning, as well as execution, continues throughout the project.

So far in the text, you have learned the detailed activities of Core Process 3 (“Discover and understand details”) and Core Process 4 (“Design system components”). This chapter discusses the activities of Core Processes 1 and 2. We elected to discuss the concepts of project management until after you had some experience in the details of project development work. In other words, your ability to understand the need and importance of project management is enhanced by having experience with the kind of work that must be managed. In later chapters, you will learn the details of the final two core processes.

FIGURE 11-5 Activities of Core Process 1



Core Process 1 is probably the most critical process for project success. As was noted earlier in the chapter, establishing such things as strong executive support, clear business case and direction, and effective planning is critical to project success. These important factors are identified and resolved during the activities of Core Process 1. **Figure 11-5** highlights the four activities associated with Core Process 1.

■ Identify the Problem

Information systems development projects are initiated for various reasons, including (1) to respond to an opportunity, (2) to resolve a problem, and (3) to respond to an external directive.

Most companies are continually looking for ways to increase their market shares or open up new markets. One way they create opportunities is with strategic plans—short term and long term. In many ways, planning is the optimal way to identify new projects. As the strategic plans are developed, projects are identified, prioritized, and scheduled.

Projects are also initiated to resolve immediate business problems. Such projects can be initiated as part of a strategic plan, but they are more commonly requested by middle managers who want to take care of some difficulty in the company's operations. Sometimes, these needs are so critical that they are brought to the attention of the strategic planning committee and integrated into the overall business strategy. At other times, an immediate need can't wait, such as a new sales commission schedule or a new report needed to assess productivity. In these cases, managers of business functions will request the initiation of individual development projects.

Finally, projects are initiated to respond to outside directives. One common version of this is legislative changes that require new information gathering and reporting—for example, changes in tax laws and labor laws. Legislative changes can also expand or contract the range of services and products that an organization can offer in a market. For example, recent changes in health insurance with the Affordable Care Act have required insurance companies to modify existing systems and add new systems to interface with government systems and the Health Insurance Marketplace.

Identifying and carefully defining the problem is a critical activity for a successful project. The objective is to ensure that the new system actually meets the business need. The purpose is to precisely define the business problem and determine the scope of the new system. This activity defines the target you want to hit.

If the target is ill defined, all subsequent activities will lack focus. For example, a request might be made for a system that would “keep track of salesperson commissions.” Without knowing more about the context surrounding this request, a system could be built that only records the commissions, ignoring the complexities of tax reporting, internal-versus-outside salespersons, deferred commissions, complex relationships, shared commissions, and so forth. Thus, even though all the specifications may not be defined in this initial activity, enough defining needs to be done to understand the implications of the required solution.

System Vision Document a document to help define the scope of a new system

An effective way to define the problem is to develop a **System Vision Document**, which was introduced in Chapter 1. There are three components to this document: the problem description, the anticipated business benefits, and the system capabilities.

business benefits the benefits that accrue to the organization; usually measured in dollars

The first task in developing a System Vision Document is to review the business needs that initiated the project. If the project was initiated as part of the strategic plan, then the planning documents need to be reviewed. If the project originated from departmental needs, then key users need to be consulted to help the project team understand the business need. From this task, a brief problem description is developed. As these needs are identified, the team also develops a detailed list of the expected business benefits. The list of **business benefits** contains the results that the organization anticipates it will accrue from a new system. Business benefits are normally described in terms of the specific results that can change the financial statements, either by decreasing costs or increasing revenues.

system capabilities the required capabilities of a new system; part of a System Vision Document

As the business benefits are being identified, the project team will identify the new system’s specific capabilities to support the realization of these benefits. The objective of this task is to define the scope of the problem in terms of the requirements for the information system. This scoping statement, as defined by a list of **system capabilities**, helps identify the size and complexity of the new system and the project that will be required.

Members of the development team, working with the users and the client, combine these three components—the problem description, the business benefits, and the system capabilities—into a System Vision Document. **Figure 11-6** presents RMO’s System Vision Document. Note the differences between the business benefits and the system capabilities. The business benefits focus on the financial benefit to the company. The system capabilities focus on the system itself. The benefits are achieved through the capabilities provided by the system.

RMO’s existing CSS (Customer Support System) was built under a tight deadline, and the company recognized that it would have a fairly short life. There were still many things to learn about Web marketing, but the existing CSS will help the company define the requirements for its CSMS.

■ Quantify Project Approval Factors

The first activity produced a high-level overview document that identified the need for a new system. However, that document alone may not be adequate to receive approval and funding. During this second activity, the project team, working with the users, will attempt to define more precisely the scope and impact of the project.

The objective is to provide sufficient justification so funds will be released and the project can start. Sometimes, the need is so great or so obvious that project approval is almost automatic. In other situations, it may be necessary to prepare a thorough cost/benefit analysis. These criteria must frequently be considered to obtain project approval:

- The estimated time for project completion
- The estimated cost for the project and system
- The anticipated benefits from the deployment of the new system

FIGURE 11-6 System Vision Document for RMO's CSMS

Consolidated Sales and Marketing System System Vision Document



Problem Description

Sales and marketing on the Web has changed drastically since the CSS was built. Customers are more sophisticated, and they are used to catalog and sales systems that are easy to use and provide many services, such as one-click ordering, deferred-purchase tracking, simplified searches, and comparison shopping. In addition, research has shown that sales increase dramatically when social media marketing tools are combined with basic sales functionality. Hence, the new CSMS is needed not only to respond to today's competition but also to launch RMO into today's world of social media and mobile computing. The longer RMO delays in starting this project, the more opportunities it misses.

System Capabilities

This document identifies the required system capabilities at a high level. Later documents will specify the detailed requirements. These capabilities are required:

- Provide a shopping cart capability.
 - Support customer sales with high automation (one-click, etc.).
 - Recommend related product purchases and comparison shopping.
 - Allow customer ratings and recommendations.
 - Include "friend" network capability.
- Include comprehensive order fulfillment.
 - Support multiple and split-order shipping and tracking.
 - Support back-ordering and tracking.
 - Allow customer comments and feedback.
- Provide customer account and billing capability.
 - Provide individualized customer accounting.
 - Support electronic billing and many electronic payment methods.
 - Accumulate customer "points" and allow transfer and sharing.
- Include marketing functions for promotions and specials.
 - Provide flexible promotions and sales.
 - Accumulate and track "points" from suppliers directly to customers.
 - Interface with social marketing media for advertising and social marketing activities.
 - Support mobile devices for social marketing and sales.

Business Benefits

The primary business benefit of these capabilities will be to increase sales by connecting with customers and improving the customer experience. The specific benefits include:

- Increasing the size of customer purchases
- Increasing the frequency of customer purchases
- Increasing customer satisfaction
- Increasing product recommendations from customers to friends
- Attracting new customers through recommendations and social marketing
- Building customer loyalty with recommendations and service
- Increasing speed of product availability
- Eliminating shipping delays and outages

At this time, these will only be rough estimates. In the traditional predictive approach to system development, estimates were often made with a considerable amount of detail. However, the estimates were frequently far off the mark. The problem was, of course, that with most new systems, the team was

venturing into unknown needs, requirements, and technologies. With the more adaptive approaches, the stakeholders recognize that the requirements are unknown and that it is more important to monitor and control scope, cost, and schedule than to try to make detailed estimates.

■ The Estimated Time for Project Completion

During Core Process 2 (“Plan and monitor the project”), a more detailed project schedule is created. During project initiation, there usually isn’t enough known about the project to create a schedule. But there is nevertheless a need to estimate the project’s completion date, even though this is one of the hardest things to do.

Sometimes, there are business constraints that dictate the completion of the project. For example, new legislative requirements may affect the deployment date, such as with the Affordable Care Act deployment date. Of course, it is always risky to assume that setting a target date actually gives the development team sufficient time to complete the project. A window of opportunity may also provide a powerful motivation to complete a project at a specific time. These considerations should be made manifest and considered in the project approval and project planning processes.

The major inputs toward estimating the project completion date are the scoping document and the amount of effort required to develop the listed requirements. As indicated earlier, it is difficult to make an estimate with any degree of accuracy. At this early point in the project, gross estimates of team size and time frame are usually the best that can be achieved. For a predictive approach, the list of requirements can serve as the starting point for estimating the effort required to define and develop a particular function. For an adaptive approach, the same information can be used to estimate the number of iterations required and the size and number of teams working on the various subsystems.

Figure 11-7 shows an example of a time estimate document for RMO.

For RMO, the development of the time estimate was a one-day exercise. Because the project didn’t yet have approval or funding, neither a project manager nor any systems analysts had been assigned to the project. However, a project manager had been assigned to obtain approval, and two systems analysts were assigned to help him. These three experienced technical people met for four hours with the key users from the various RMO departments. The object of these meetings was to build a comprehensive list of all the functional requirements from each department. After the meetings, the group met again to organize this list of requirements into groupings that could be assigned to various iterations for the development of the software.

FIGURE 11-7 Project completion date estimate for the CSMS project

Time Estimate for the New CSMS Project			
Subsystem	Functional requirements	Iterations required	Estimated time
Sales subsystem*	15	5	20 weeks
Order Fulfillment subsystem*	12	5	20 weeks
Customer Account subsystem**	10	4	15 weeks
Marketing subsystem**	6	3	13 weeks
Reporting subsystem**	7	3	12 weeks
Total development time (2 teams)			40 weeks
Final hardening and acceptance testing		2	8 weeks
Total project time			48 weeks

*Assigned to Tiger team

**Assigned to Cougar team

FIGURE 11-8 Summary of development costs for CSMS

Summary of Development Costs for CSMS	
Expense category	Amount
Salaries/wages (includes benefits costs) (1 PM, 8 analysts, 1 support)	\$936,000.00
Equipment/installation	\$308,000.00
Training	\$78,000.00
Facilities	\$57,000.00
Utilities	\$97,000.00
Travel/miscellaneous	\$87,000.00
Licenses	\$18,000.00
Total	\$1,581,000.00

© Cengage Learning®

An assumption that was made by the director of new development was that there would be two subteams of four people each allocated to this project. As indicated in Figure 11-7, the time estimate for this project is 48 weeks from the date it begins.

■ The Estimated Cost for the Project and System

The estimated costs of developing the new CSMS are shown in **Figure 11-8**. By far, the largest cost item in the project's budget is the salaries of the project team. Other cost elements include the cost of the new computers; training for the users; offices, facilities, and utilities for the project team; travel expenses for the project team to do site visits; and software licenses. As you can see, this estimate is a little over \$1.5 million.

After the system has been put into production, there will be annual operating costs, as shown in **Figure 11-9**. The largest cost is for a hosting service to provide some of the equipment, the connection to the Internet, and server administration services. These estimated costs were based on RMO using a hosting service to provide the equipment, the connection to the Internet, and server administration. The project team estimated about \$13,000 a month for those expenses, which is enough for 15 large managed servers. This appeared to be more than adequate depending on the traffic volume. Other costs were for one full-time programmer and two help desk personnel.

■ The Anticipated Benefits from the Deployment of the New System

The System Vision Document identifies the anticipated business benefits of the new system. In this task, you analyze those business benefits and provide an estimate of their value to the organization. This value becomes part of the total decision criteria. Obviously, the dollar amount associated with these savings or revenues must be estimated by the client. It isn't the project manager's job to

FIGURE 11-9 Summary of estimated annual operating costs for CSMS

Summary of Estimated Annual Operating Costs for CSMS	
Recurring expense	Amount
Connectivity/hosting	\$156,000.00
Programming	\$75,000.00
Help desk	\$90,000.00
Total	\$321,000.00

© Cengage Learning®

predict the value of business benefits. However, the project manager can help the client identify categories of potential benefits. Typical areas of increased revenue or cost reduction benefits include:

- Opening up new markets with new services, products, or locations
- Increasing market share in existing markets
- Enhancing cross-sales capabilities with existing customers
- Reducing staff by automating manual functions or increasing efficiency
- Decreasing operating expenses, such as shipping charges for “emergency shipments”
- Reducing error rates through automated editing or validation
- Reducing bad accounts or bad credit losses
- Reducing inventory or merchandise losses through tighter controls
- Collecting receivables (accounts receivable) more rapidly

The project team at RMO worked with the vice president of sales and marketing to identify benefit areas and estimate a value for each one. This size of an investment and ongoing expense was going to require board approval within RMO. The board will want to know what the benefits of the new system will be and what the return on the investment will be. One of the difficulties for RMO is to determine how to assign a value to a benefit. A typical question might be “Do we assign the value of all our sales given that this system is needed to stay competitive in the marketplace? Or do we assign only the value of the increased sales we expect to get from marketing and higher volume?” If sales will drop because RMO becomes less competitive in the marketplace, the total sales value could be used. However, if the existing system is good enough to maintain a good client base, then only the increased sales should be used. These kinds of decisions are made by the client, not the project team. In this case, the vice president of sales and marketing at RMO decided to use a more conservative estimate. **Figure 11-10** summarizes the estimates he generated.

Many organizations like to compare the estimated costs with the anticipated benefits to calculate whether the benefits outweigh the costs. This process is called a **cost/benefit analysis**. Companies use a combination of methods to measure the overall benefit of the new system. One popular approach is to determine the **net present value (NPV)** of the new system. The two concepts behind net present value are (1) that all benefits and costs are calculated in terms of today’s dollars (present value) and (2) that benefits and costs are combined to give a net value. The future stream of benefits and costs are netted together and then discounted by a factor for each year in the future. The discount factor is the rate used to bring future values back to current values. Online Chapter C, “Project Management Techniques,” includes instructions on how to calculate estimated benefits using net present value and other financial measures.

cost/benefit analysis process of comparing costs and benefits to see whether investing in a new system will be beneficial

net present value (NPV) the present value of dollar benefits and dollar costs of a particular investment

FIGURE 11-10 *Estimated annual benefits for CSMS*

Estimated Annual Benefits for CSMS	
Benefit or cost saving	Amount
Recapture/prevention of lost sales	\$200,000.00
Increase sales to existing customers	\$300,000.00
Sales to new customers	\$350,000.00**
Increased efficiency in order processing	\$50,000.00
Reduction of data center and equipment costs because of hosting	\$146,000.00
Total	\$1,046,000.00

**plus 8% annual growth

FIGURE 11-11 Five-year cost/benefit analysis for CSMS

RMO Cost / Benefit Analysis for CSMS							
Category	Year 0	Year 1	Year 2	Year 3	Year 4	Year 5	
1 Value of benefits		\$1,046,000	\$1,074,000	\$1,104,240	\$1,136,899	\$1,172,171	
2 Development costs	-\$1,581,000						
3 Annual expenses		-\$321,000	-\$321,000	-\$321,000	-\$321,000	-\$321,000	
4 Net benefit/costs	-\$1,581,000	\$725,000	\$753,000	\$783,240	\$815,899	\$851,171	
5 Discount factor (6%)	1.0000	0.9434	0.8900	0.8396	0.7921	0.7473	
6 Net present value	-\$1,581,000	\$683,965	\$670,170	\$657,608	\$646,274	\$636,080	
7 Cumulative NPV	-\$1,581,000	-\$897,035	-\$226,865	\$430,743	\$1,077,017	\$1,713,097	
8 Payback period	2 years +	226,865 / (226,865 + 430,743) = .35			or 2 years + 128 days (.35*365)		

Source: Microsoft Corporation

Figure 11-11 shows a copy of the NPV calculation done for RMO's new CSMS. There are various techniques for calculating the NPV of a given investment. In this example, Year 0 represents the development period prior to the deployment of the system. The annual benefits for each year are extended across the top row. The development costs are shown on the second row. Annual expenses are shown on the third. Those three rows are combined in the fourth row to give the net benefits and costs. The fifth row shows the discount value given a 6 percent discount rate. The sixth row is the product of the fourth and fifth rows and represents the net value in terms of today's dollars (i.e., the NPV). The seventh row shows a cumulative total of annual NPVs.

In Figure 11-11, the numbers in the seventh row eventually change from negative to positive. The point in time when that happens is called the **break-even point**. The length of time before the break-even point is reached is called the **payback period**. The payback period occurs in the year that the cumulative value goes positive. To calculate it, first take the last year that the cumulative value is negative—in this case, Year 2. Add to that year the number of days in the following year (in this case, Year 3) that it takes for the cumulative value to go positive. The method for doing that is to take absolute values of the ending value in Year 2 divided by the sum of the absolute values for the end of Year 2 and Year 3—in this case, 226,865 divided by (226,865+430,743). Here, that calculation indicates that the cumulative value goes positive after 35 percent of the year has passed. Multiply .35 times the 365 days in the year to get 128 days into Year 3. Many companies require a payback period of two to three years on new software.

The previous cost/benefit calculation depends on an organization's ability to quantify the costs and benefits. If it can indeed estimate a dollar value for a benefit or a cost, the organization treats that value as a **tangible benefit** or cost. However, in many instances, an organization can't measure some of the costs and benefits to determine a value. Never discount the importance of ascertaining the "behind the scenes" reasons for a project. There may be political reasons for or against the project that override all other feasibility analyses. If there is no reliable method for estimating or measuring the value, it is considered an **intangible benefit**. In some instances, the importance of the intangible benefits far exceeds the tangible costs—at least in the opinion of the client, who pursues developing the system even though the dollar numbers don't indicate a good investment.

Examples of intangible benefits include:

- Increased levels of service (in ways that can't be measured in dollars)
- Increased customer satisfaction (not measurable in dollars)
- Survival
- Need to develop in-house expertise (such as a pilot program with new technology)

break-even point the point in time at which dollar benefits offset dollar costs

payback period the time period during which the dollar benefits offset the dollar costs

tangible benefit a benefit that can be measured or estimated in terms of dollars

intangible benefit a benefit that accrues to an organization but that can't be measured quantitatively or estimated accurately

Examples of intangible costs include:

- Reduced employee morale
- Lost productivity (the organization may not be able to estimate it)
- Lost customers or sales (during some unknown period of time)

■ Determine Project Risk and Feasibility

Project risk and feasibility analysis verifies whether a project can be started and completed successfully. Because each project is a unique endeavor, every project has unique challenges that affect its potential success.

The objective of this activity is to identify and assess the potential risks to project success and to take steps to eliminate or at least ameliorate these risks. They should be identified during the project approval process so all stakeholders are aware of the potential for failure. The team can also establish plans and procedures to ensure that those risks don't interfere with the success of the project. Generally, the team assigns itself these tasks when confirming a project's feasibility:

- Determine the organizational risks and feasibility.
- Evaluate the technological risks and feasibility.
- Assess the resource risks and feasibility.
- Identify the schedule risks and feasibility.

■ Determine Organizational Risks and Feasibility

Each company has its own culture, and any new system must be accommodated to that culture. There is always the risk that a new system departs so dramatically from existing norms that it can't be successfully deployed. The analysts involved with feasibility analysis should evaluate organizational and cultural issues to identify potential risks for the new system. Such issues might include:

- Substantial computer phobia
- A perceived loss of control on the part of staff or management
- Potential shifting of political and organizational power due to the new system
- Fear of change of job responsibilities
- Fear of loss of employment due to increased automation
- Reversal of long-standing work procedures

It isn't possible to enumerate all the potential organizational and cultural risks that exist. The project management team needs to be very sensitive to the reluctance within the organization to identify and resolve these risks.

After identifying the risks, the project management team can take positive steps to counter them. For example, the team can hold additional training sessions to teach new procedures and provide increased computer skills. Higher levels of user involvement in developing the new system will tend to increase user enthusiasm and commitment.

■ Evaluate Technological Risks and Feasibility

Generally, a new system brings new technology into the company, even state-of-the-art technology. Other projects use existing technology but combine it into new, untested configurations. If an outside vendor is providing a capability in a certain area, the client organization usually assumes the vendor is an expert in that area. However, even an outside vendor may find the requested level of technology too complicated.

The project management team needs to carefully assess the proposed technological requirements and available expertise. When these risks are identified, the solutions are usually straightforward. The solutions to technological risks

include providing additional training, hiring consultants, or hiring more experienced employees. In some cases, the scope and approach of the project may need to be changed to ameliorate technological risk. The important point is that a realistic assessment will identify technological risks early, making it possible to implement corrective measures.

■ Assess Resource Risks and Feasibility

The project management team must also assess the availability of resources for the project. The primary resource consists of team members. Development projects require the involvement of systems analysts, system technicians, and users. Required people may not be available to the team at the necessary times. An additional risk is that people assigned to the team may not have the necessary skills for the project. After the team is functioning, members may have to leave the team. This threat can come either from staff who are transferred within the organization if other special projects arise or from qualified team members who are hired by other organizations. Although the project manager usually doesn't like to think about these possibilities, skilled people are in short supply and sometimes do leave projects.

The other resources required for a successful project include adequate computer resources, physical facilities, and support staff. Generally, these resources can be made available, but the schedule can be affected by delays in the availability of these resources.

■ Identify Schedule Risks and Feasibility

The development of a project schedule always involves high risk. Every schedule requires many assumptions and estimates without adequate information. For example, the needs (and, hence, the scope) of the new system aren't well known. Also, the time needed to research and finalize requirements has to be estimated. The availability and capability of team members aren't completely known.

Another frequent risk in developing the schedule occurs when upper management decides that the new system must be deployed within a certain time. Sometimes, there is an important business reason for setting a fixed deadline, such as RMO's need to complete the CSS in time for online ordering over the holidays. Similarly, universities require the completion of new systems before key dates in the university schedule. For example, if a new admissions system isn't completed before the admissions season, then it might as well wait another full year. In cases like these, schedule feasibility can be the most important feasibility factor to consider.

If the deadline appears arbitrary, the tendency is to create the schedule to show that it can be done. Unfortunately, this practice usually spells disaster. The project team should create the schedule without any preconceived notion of required completion dates. After the schedule is completed, comparisons can be done to see whether timetables coincide. If not, the team can take corrective measures, such as reducing the scope of the project, to increase the probability of the project's on-time completion.

One objective of defining milestones and iterations during the project schedule is to permit the project manager to assess the ongoing risk of schedule slippage. If the team begins to miss milestones, the manager can possibly implement corrective measures early. Contingency plans can be developed and carried out to reduce the risk of further slippage.

■ Review with Client and Obtain Approval

As mentioned earlier, the amount of expenditure for the RMO project required board approval. However, before a presentation could be given to the board, RMO's executive committee needed to understand and agree to the project.

A project of this size has major impacts on all areas of the company. The departments, such as Sales and Marketing, will be directly impacted. They will have to allocate staff and resources to help in defining the requirements, developing test cases, and testing the new system as it is developed. In other words, the people in this department will have extra duties for the next 12 months or so. Even departments not directly involved will need to support this heavy development activity, perhaps tightening their budgets. In any event, it is always good policy to get the approval and support of the entire company. This process starts by making presentations to the senior executives of RMO. Often, a project manager will be asked to make the presentation or at least be present to answer questions.

After the executive committee approves the project, it goes to the board. After board approval, the IT Department begins to assign full-time resources to the project. It is also a good idea at this point to have a company-wide memo or meeting to mark the beginning of this major activity. If the entire company knows that all the executives are supporting it and requesting cooperation, the project will proceed much more smoothly.

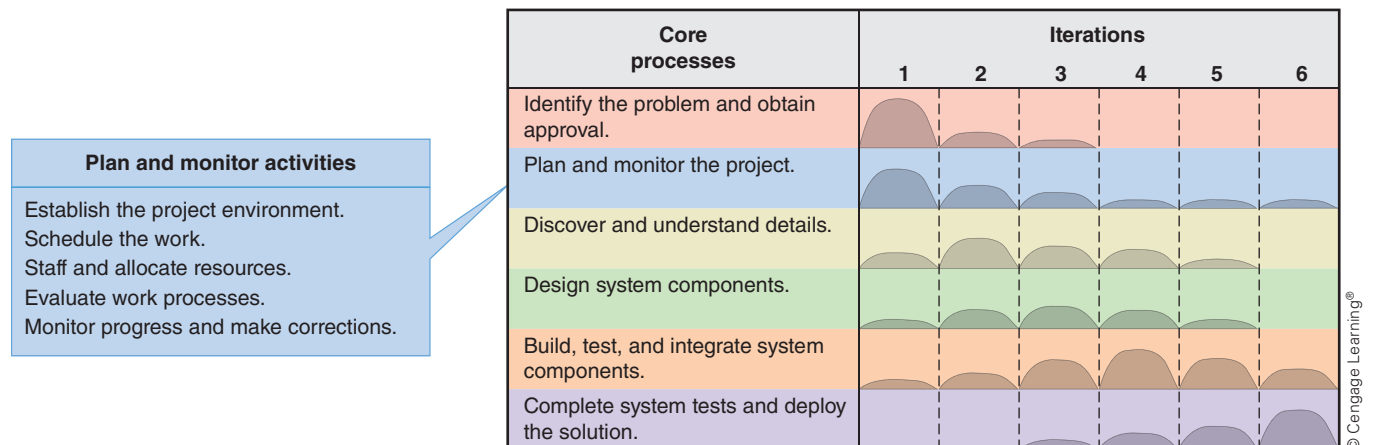
■ Activities of Core Process 2: Plan and Monitor the Project

This core process lasts throughout the entire project. A major planning effort occurs immediately after the project is approved. Ongoing planning and project monitoring continue during all project iterations. Not only must each iteration be planned as it starts, but progress must continually be monitored and corrective actions may be required. **Figure 11-12** illustrates by the height of the effort curve in each iteration that planning and monitoring activities must be an integral part of every project iteration. The specific activities associated with this core process are also listed in Figure 11-12. The following sections discuss each of these activities individually.

■ Establish the Project Environment

So far, this text has discussed different types of projects, such as predictive and adaptive projects, as well as tools, techniques, and methodologies to use with these different types of projects. It has also discussed such concepts as ceremony, project reporting, stakeholders, user participation, and the project team work environment. All these elements must be put in place as the project gets under way.

FIGURE 11-12 *Activities of Core Process 2*



Some of these decisions will already have been made based on the organization's standard policies and procedures. Others will be decided during the approval process. In any case, the project manager must ensure that the project's parameters and the work environment are finalized so the work of the project can proceed without roadblocks or delays. There are important project structure considerations that must be addressed as the project gets under way. For example, what kind of communication processes will be needed to keep the team and external stakeholders informed about what is going on? In addition, the members of the project team all need computers and integrated development environments (IDEs) and other tools to do their work. Of course, specific procedures about how the project team meets with the users, how they write code, and how they submit code for acceptance also need to be finalized. The following sections discuss three important considerations:

- Recording and communicating—internal/external
- Work environment—support/facilities/tools
- Processes and procedures

■ Recording and Communicating—Internal/External

The project manager and project team members will be involved in all types of meetings where decisions will be made and information developed. Determining what is important and how to record this information need to be set out in specific project procedures. The other critical issue with information is what, how, how frequently, and to whom this information needs to be disseminated. One of the first tasks for a project manager on a new project is to establish the procedures and guidelines for how to handle the project's information.

A critical success factor for IT projects is to have the support of the organization's executives and other key stakeholders. A good project manager understands this need and structures his or her project so he or she communicates frequently, with the appropriate detail, to each of his or her stakeholders. Figure 11-2 identified the various stakeholders and participants in a project. Some of these stakeholders will be integrally involved with the project. Other stakeholders will be only marginally involved, receiving periodic status reports. The client stakeholders (the ones paying the project costs) will need to be kept aware of the project's status and of any difficulties or delays. A stakeholder analysis helps identify all those persons who have an interest in the project and defines what information they will want and need concerning the project. Generally, we refer to this as external reporting of project information.

Maintaining project information can be done via electronic means. Spreadsheets, e-mails, newsletters, and list servers all provide ways to maintain, collect, and distribute information. Once the electronic systems are set up, they will often take care of themselves. Project information can be published to a Web site so everyone can view it.

Another type of project-tracking tool, sometimes called a *project dashboard*, allows all types of project information to be posted and viewed by Web browsers. There are many available application systems that will extract the project data and illustrate it in different formats. Many of these systems are flexible and allow the project manager to configure the data as well as the look and feel of the project dashboard. **Figure 11-13** is an example of a project dashboard system that allows easy access to project information. Each of the tiles on the dashboard is a hot link and will open up a new window with more details, charts, and graphs.

The members of the project team also need to have mechanisms in place to communicate among themselves and document project decisions. This is an entirely different type of information—information about the system under development. For example, during analysis activities, the project team documents the results of user meetings by using various means, such as writing use

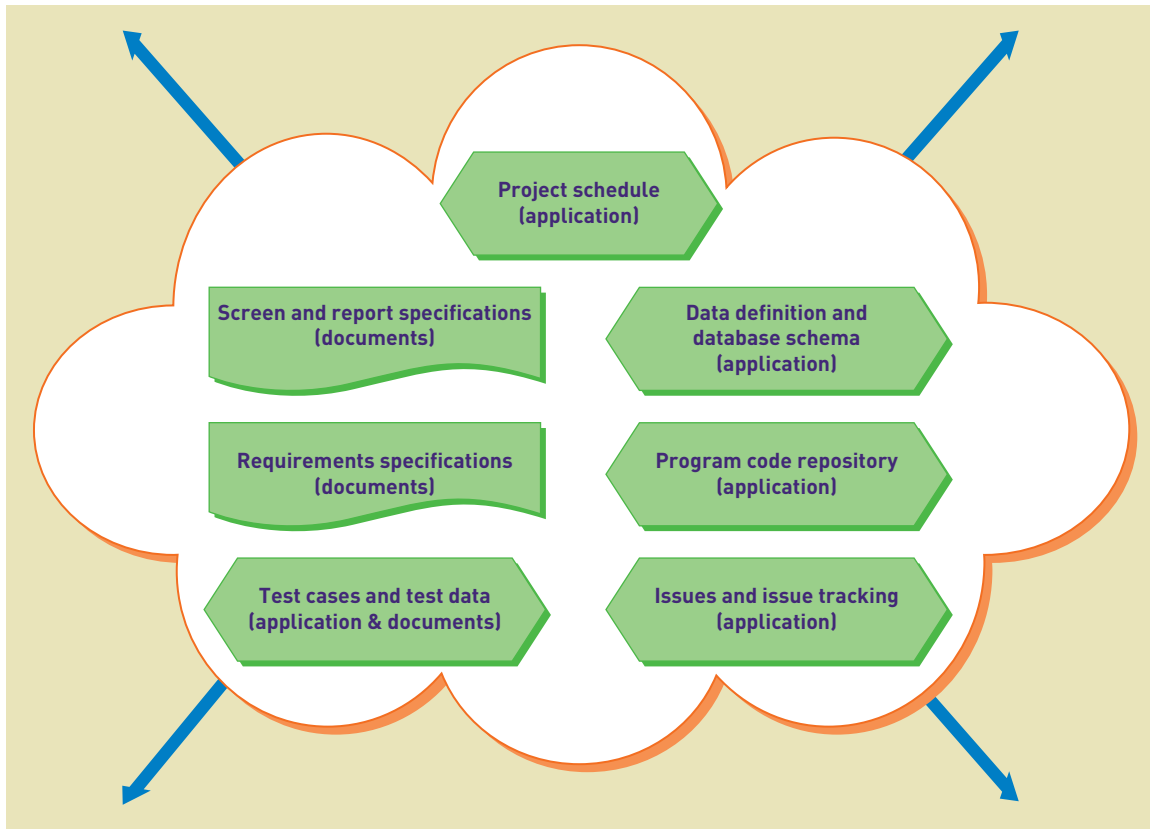
FIGURE 11-13 Sample dashboard showing project information and status

Conference Registration System									
Planned finish	Dec 2015	Urgent issues	3	Tasks this iteration	45				
Planned cost	\$1,680,000	Open issues	22	Tasks this week	15				
Time expended	18%	Assigned	15	Open	35				
Deliverables completed	4/35	Unassigned	7	Unassigned	22				
Current iteration	4/15	Closed issues	137	Assigned	13				
Active tasks	5	Identified risks	4	Slipped – urgent	3				
				Slipped – routine	7				
				Completed tasks	10				
Project Overview		Issues & Risks		Tasks					
Iterations	11	Status Report	Total	\$1,680,000	Status	short 1	Total	87	
Current	4	Domain Model	Expended	\$486,000	Team ldrs	3 full	This iteration	14	
This iteration		User Stories	Labor	\$397,000	Prog/anal	7 full	Completed	4	
Length	42 d	Process Improve	Servers	\$27,000	Prog/anal	2 prt	Slipped	2	
Left	21 d		Contractors	\$52,000	Tech supp	1 prt	Deliverables	3	
			Facilities	\$10,000	User supp	4prt	Finished	1	
Iteration Schedule		Documents		Budget		Team Staff		Milestones	

© Cengage Learning®

case descriptions. During design, information also needs to be recorded and distributed among members of the team as appropriate. During testing, when errors are found, they must be documented and assigned to programmers to be fixed. Finally, the entire recording and communication requirement is often made more critical by members of the project team (as well as the users) being located at various sites around the globe. Figure 11-14 illustrates some of the

FIGURE 11-14 System information stored in data repositories



© Cengage Learning®

information that may need to be captured and maintained. The data repository in the figure usually consists of many different types of data structures and storage techniques—from wikis to databases to issue-tracking systems.

There is one caveat related to recording and communicating. With traditional predictive projects, the tendency was to create reams and reams of documentation. As you learned in the previous chapter, adaptive projects that use the Agile philosophy emphasize code over documentation. A novice project manager may interpret that to mean that no documentation is required. However, even with an Agile approach, the basic user definitions need to be documented for later verification. It isn't uncommon during programming for a programmer to have to refer back to notes and models to remember the exact details and decisions of a particular requirement. An experienced project manager knows the right amount of documentation so the project isn't overloaded with overhead but that critical decisions are recorded.

It should be obvious that a comprehensive recording and communication scheme needs to be put in place. Fortunately, in today's connected world, there are many tools available so external and internal communication can be done easily. With so many electronic tools, all project information should be available online and accessible to all stakeholders. In fact, with the use of wikis, it is now common to allow many team members and even users to assist in the recording and updating of critical project information.

The CSMS team wanted to maintain its project information in digital format and have it available to all stakeholders, including team members, users, the client, and the members of the steering committee. RMO is a very open shop guided by the philosophy that information should be widely distributed. **Figure 11-15** shows all the tools that the CSMS project team uses to communicate and capture information. The core team members had previously worked on several Agile projects, so they had learned there is a correct balance of

FIGURE 11-15 *Electronic digital repositories of information for CSMS*

Electronic Digital Repositories		
Information captured	Electronic tools	Who can update/view
User definitions and functions User documents	Forum software Document server Scanners	Analysts, users/all
Screens and reports layouts	Web design tools Visio PowerPoint/Keynote	Analysts, users/all
Design specifications and diagrams	Wiki software Visio	Analysts/all
Issues and outstanding problems	Issue-tracking software	Analysts, users/all
Program code	Apache subversion (SVN)	Analysts
Project schedule	Microsoft project	Analysts/all
Project status and information	Forum software	Analysts, users/all
Daily team coordination meeting	Video laptop conferencing	Project team
Distributed team communication	IM chat with video	Project team
Project update newsletter	Blog software	Project manager/all

documentation—not too much but enough to be able to trace key decisions and requirements. Barbara Halifax, the project manager, wanted to ensure the tools were in place so it was easy to record information when it was prudent to do so.

User documents, such as sample invoices, were scanned and placed in a document repository. User functional definitions were recorded in a forum system. Using a forum allowed team members and users to update it when key issues were discussed and needed to be remembered. Sample screen and report layouts were either sketched out or drawn with Visio or Keynote. Hand sketches were often scanned and saved. Most design decisions and specifications went right into the program code and weren't documented. However, some decisions were global, and those were captured in a wiki.

Each day, the project team had a “stand up” meeting—a short coordination meeting. Most of the team members were in the Park City Center, but some users were assigned to the team from other locations. Sometimes, team members were visiting user sites and therefore not available, and there were some team members who worked in the Salt Lake City office. Therefore, the daily meeting was conducted as a videoconference call using Skype, with each person using his or her webcam and personal computer. The meeting normally lasted about 15 minutes.

Finally, there was some discussion of sending out a biweekly newsletter about the progress on the project. Barbara felt that it was important for the entire company to stay informed about the project to encourage their enthusiasm and support. However, instead of a printed newsletter, she opted to do it in the form of a blog. All users were invited to sign up with an RSS feed to keep informed about the project's progress.

■ Work Environment—Support/Facilities/Tools

Although the work environment may relate more to the work processes of the project team, the project manager must ensure that it is adequate to allow the project team to work productively. There are five major components of the work environment:

- Personal computer(s) and/or workstation(s)
- Personal development software and tools
- Development server with repositories, sandboxes, and communication tools
- Office space, conference rooms, and equipment, including printers, scanners, and projectors
- Support staff

Most important, of course, is the computer equipment and other hardware that the team will need. Obviously, each developer will need his or her own computing configuration, which may consist of multiple computers or monitors. Other important hardware includes the development servers, printers, and internal development network. If the team is distributed, video cameras and projectors may be necessary to conduct distributed team meetings. Along with the hardware, resources must be made available to administer things such as the development server.

Related to the hardware is the computer software and other tools. Software tools can get quite elaborate—from stand-alone IDE tools to modeling software to code repository software. The development server, with its environment and software, must also be configured and deployed. The server may be set up as a virtual server or as a stand-alone computer. Applications include such things as code repository, issue-tracking application, testing system, and the project dashboard.

Along with the hardware and software, a work configuration must be provided for each developer, with logon permissions, sandbox environments, repository access, and so forth. The final two components are the office space and other facilities that may be needed. This will include access to conference rooms,

presentation equipment, and maybe even transportation vehicles. Finally, the productivity of the team members is always enhanced when adequate support staff is available to take care of myriad details that always accompany an active project.

■ Processes and Procedures

The final major set of decisions has to do with the project's internal processes and procedures. Earlier, you learned about a project's level of formality. Larger projects require more elaborate reporting processes and meeting schedules. When there are many people involved, coordination of activities becomes critical. Procedures include:

- Reporting and documentation—What is done? How is it done? Who does it?
- Programming—Single or pair programming? How is work assigned? By whom?
- Testing—Programmer tests or user tests? How do you mark items ready for testing?
- Deliverables—What are they? How and when are they handed over to users? How are they accepted?
- Code and version control—How is the code controlled to prevent conflicts? How do you coordinate bug fixing with new development? How and when are deliverables released?

■ Schedule the Work

Scheduling the work is necessary for any size or type of project. However, the techniques used can vary widely depending on the type of project. For predictive, highly controlled projects, a detailed and complete schedule that covers the entire project is usually built. Again, these kinds of schedules only work because the software to be built is well understood. However, even in those projects with detailed and comprehensive schedules, accommodation is required as things change during the life of the project. At the other end of the spectrum, small Agile projects sometimes don't even have a project schedule, with the team members being responsible for scheduling their own work. Coordination is accomplished by talking and keeping each other informed of what each person is working on. This is what is meant by *chaordic*.

Scheduling the work for many of today's projects lies somewhere between these two extremes. Large projects may have several independent teams of developers working on various subsystems. Even though the work between the teams is fairly independent, coordination is still required. Adaptive projects also anticipate additional requests and changes to the original scheduled tasks.

For adaptive types of projects, creating the project schedule is done throughout the life of the project. During the initial planning phase, the initial list of use cases or user stories is developed for each subsystem. The use cases are divided up and tentatively assigned to the iterations. Let us call this the **project iteration schedule**. As each iteration is begun, a detailed schedule of tasks and work to be done is developed. You saw an example of creating an iteration schedule in Chapter 1. Let us call this schedule a **detailed work schedule**, meaning that it schedules the work within an iteration. Sometime during each iteration—often as one iteration is finishing and before the next iteration begins—the project manager, with assistance from the team leaders and key users, will review and rework the project iteration schedule. During this process, the changes and any new requirements are prioritized and placed on the schedule.

Creating the project iteration schedule must take into account the total size and configuration of the solution system and the number of teams available to work on the project. Separate lists of requirements are made by each subsystem,

project iteration schedule the list of iterations and use cases or user stories assigned to each iteration

detailed work schedule the schedule that lists, organizes, and describes the dependencies of the detailed work tasks

FIGURE 11-16 *Project iteration schedule for the Sales subsystem*

Project Iteration Schedule for the CSMS Sales Subsystem		
Iteration	Time estimate	Use cases assigned to iteration
1	4 weeks	1. Search for item. 2. View detailed descriptions. 3. View rotating (3-D) images. 4. Compare item characteristics.
2	4 weeks	5. View comments and ratings. 6. Search comments and ratings for friends. 7. View accessory combinations (images). 8. Save item + accessories as "combo."
3	5 weeks	9. Add item (or combo) to shopping cart. 10. Remove item (or combo) from shopping cart. 11. Add item (or combo) to "on reserve" cart. 12. Remove item (or combo) from "on reserve" cart.
4	4 weeks	13. Check out active cart. 14. Create and process store sale. 15. Create and process phone sale.
5	3 weeks	16. Clean up, final test, harden site, tune database, etc.
Total	20 weeks	

© Cengage Learning®

and a project iteration schedule can then be made for each subsystem. Some tasks, such as designing the database, may go across all subsystems and need to be scheduled separately or be included in every subsystem list. **Figure 11-16** shows a sample project iteration schedule for the CSMS Sales subsystem. As you can see, the length of each iteration is fairly constant at around four weeks. All the identified tasks, which represent the requirements, have been assigned to iterations. In this case, we have identified five iterations.

Developing a detailed work schedule for a single iteration is a three-step process:

- Develop a work breakdown structure.
- Estimate effort and identify dependencies.
- Create a schedule by using a Gantt chart.

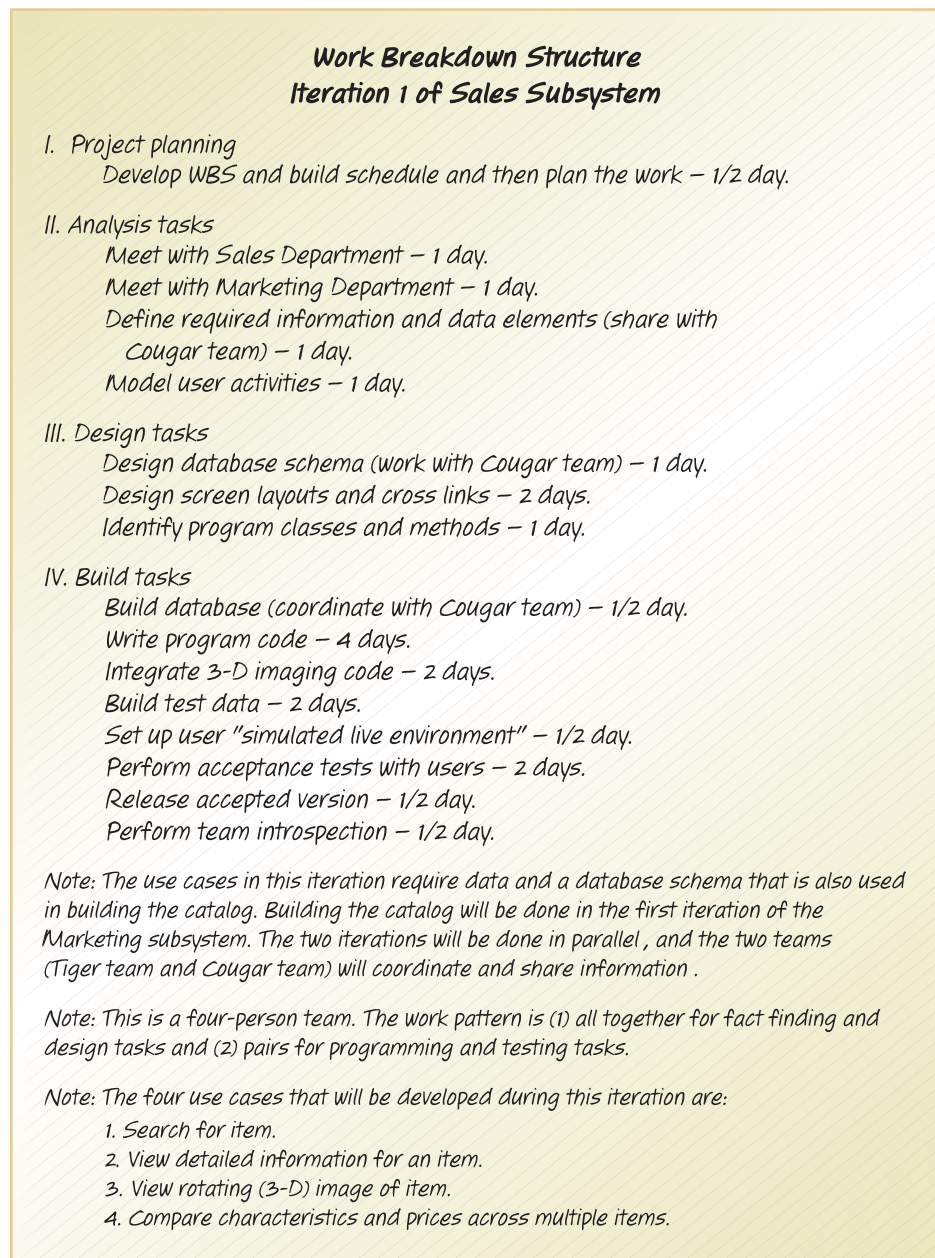
work breakdown structure (WBS)

the list or hierarchy of activities and tasks of a project; used to estimate the work to be done and to create a detailed work schedule

A **work breakdown structure (WBS)** is a list of all the required individual activities and tasks for the project. There are two general approaches for creating a WBS: by deliverable or by a timeline. The first approach identifies all the deliverables that must be completed for a given iteration. Then, the WBS identifies every task that is necessary to create each deliverable. The second approach works through the normal sequence of activities that are required for the final deliverable. Experienced developers who have worked on Agile projects understand the steps and tasks that are required to create a particular deliverable. Of course, each iteration is slightly different depending on the particular functions and deliverables that are included.

Figure 11-17 is a sample handwritten WBS for the first iteration of the Sales subsystem. The tasks have been partitioned according to the core processes Planning, Analysis, Design, and Building. In the figure, each task also has an estimate of the time required. Sometimes, two estimates are provided: the effort required and the expected duration. The effort required is given in person-days of work, and the duration is a measure of lapsed calendar time. Of course, these are related depending on the number of people working on the specific task. In **Figure 11-16**, only duration is shown; however, the time estimates assume a project team of four people.

FIGURE 11-17 Work breakdown structure for first iteration



When developing a WBS, new analysts frequently ask, “How detailed should the individual tasks be?” A few guidelines can help answer that question:

- There should be a way to recognize when the task is complete.
- The definition of the task should be clear enough so you can estimate the amount of effort required.
- As a general rule for software projects, the effort should take one to five working days.

The second step in developing a detailed work schedule for a single iteration is to determine the dependencies between the tasks and the amount of effort required for each. The most common way to relate tasks is to consider the order in which they are completed; for example, as one task finishes, the next one starts. This is called a finish-start relationship. Other ways to relate tasks include start-start relationships, in which tasks start at the same time, and finish-finish

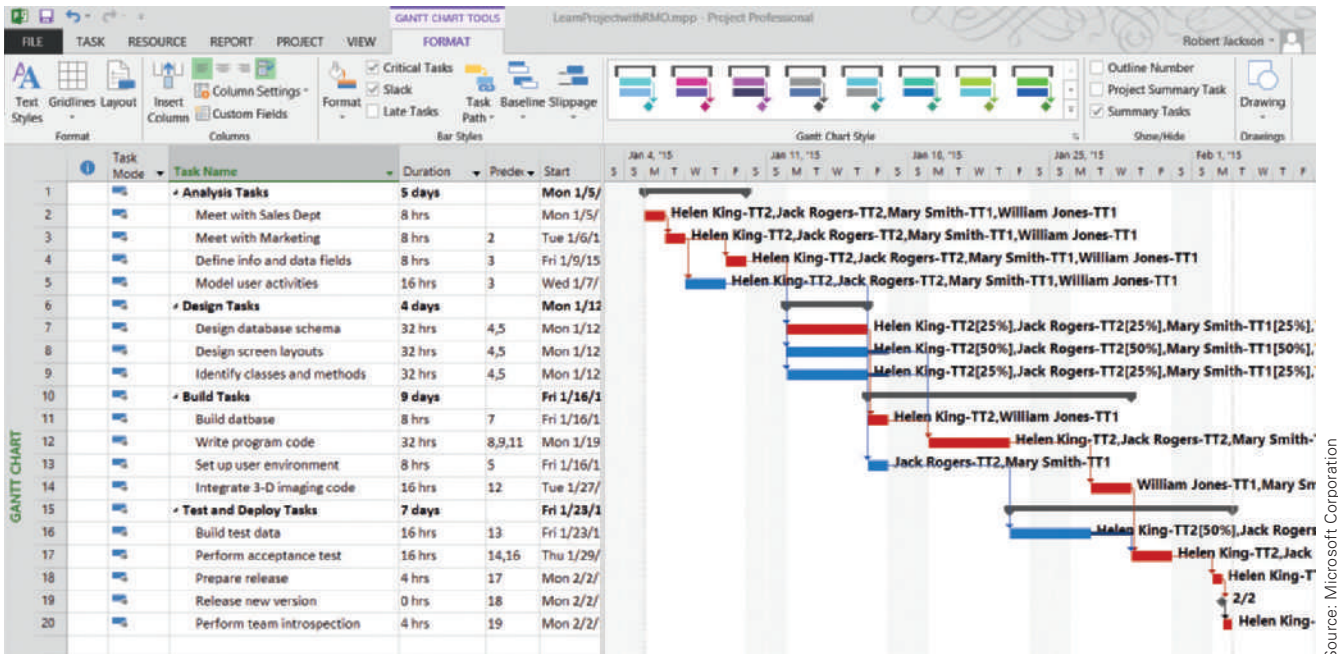
relationships, in which tasks must finish at the same time. The effort required should be the actual amount of work required to complete the task. As with the identification of the tasks in the WBS, the dependencies and effort estimates should be done by the developers who are going to actually do the work.

The third step in developing a detailed work schedule is to actually create the iteration schedule. In Chapter 1, Figure 1-10 presented a graph of the tasks involved in the first iteration of the Tradeshow System, their sequence, and the estimated calendar time to complete them. The graph was, in actuality, a simplified PERT/CPM chart. Online Chapter C provides more information about PERT charts. The other form for presenting a schedule is a bar chart that shows the activities as bars on a horizontal timeline; this is called a **Gantt chart**. A widely used tool for building Gantt charts is Microsoft Project. New versions of Microsoft Project are network enabled and provide a powerful tool to not only create schedules, but to also distribute schedule information across the organization by using HTML and the HTTP protocol so it can be viewed in a browser. The benefit of using a tool such as Microsoft Project is that the project manager can update progress easily and make that information widely available. Online Chapter C provides more detailed instructions both on Gantt charts and on the Microsoft Project management software.

Gantt chart a bar chart that portrays the schedule by the length of horizontal bars superimposed on a calendar

Figure 11-18 shows an iteration schedule from the RMO CSMS project formatted as a bar chart. In the figure, the tasks from the work breakdown structure are listed in the Task Name column and the durations are listed in the Duration column. The Predecessor column identifies dependencies between tasks. As you can see, every task except the first has at least one predecessor task, and every task except the last is a predecessor to one or more other tasks. There are various ways to document dependencies. The most common way is to show the finish of one task occurring before the start of another (FS). Other common ways are finish-finish (FF), where both must finish at the same time, and start-start (SS), where both must start at the same time. Any dependency can have a lag time, such as that shown on Line 11 of Figure 11-18. The final column documents what resources have been assigned to each task. In this example, the Tiger Team is divided into two subteams of two people each: TT1 and TT2.

FIGURE 11-18 An iteration schedule for the first iteration of the Shopping Cart subsystem



critical path a sequence of tasks that can't be delayed without causing the entire project to be delayed

The bars in Figure 11-18 illustrate the duration of each task superimposed on a calendar. The red bars indicate a critical path on the schedule. The **critical path** is defined as those tasks that must stay on schedule. If any of the critical path tasks cause a schedule slip, then the entire project is delayed. The blue bars are those tasks that aren't on the critical path. Obviously, a project manager will monitor critical path tasks quite closely. Online Chapter C gives more detailed explanations and instructions on how to use Microsoft Project to create Gantt chart schedules.

■ Staff and Allocate Resources

In an Agile project, the various teams are self-organizing. They decide how they are going to work together and assign the tasks to be done among themselves. However, the job of identifying what expertise is needed for the project and getting those people assigned to the project falls on the shoulders of the project manager. This includes finding the right people with the correct skills and then organizing and managing them throughout the project. The staffing activity consists of five tasks:

- Developing a resource plan for the project
- Identifying and requesting specific technical staff
- Identifying and requesting specific user staff
- Organizing the project team into work groups
- Conducting preliminary training and team-building exercises

Based on the tasks identified in the project schedule, the project manager can develop a detailed resource plan. In fact, the schedule and the resource requirements are usually developed concurrently. In developing the plan, the project manager recognizes that (1) resources usually aren't available as soon as requested and (2) a period of time is needed for a person to become acquainted with the project. After developing the plan, the project manager can then identify specific people and request that they become part of the team.

On small projects, members of the project team may all work together. However, a project team that is larger than four or five members is usually divided into smaller work groups. Each group will have a group leader who coordinates the tasks assigned to the group. The project manager is responsible for dividing the team into groups and assigning group leaders.

Finally, training and team-building exercises are conducted. Training may be done for the project team as a whole when such new technology as a new database or a new programming language is used. In other cases, team members who are unfamiliar with the tools and techniques being used may require individual training. The team should conduct appropriate training for technical people and users. Team-building exercises are especially important when members haven't worked together before. The integration of users with technical people is an important consideration in developing effective teams and workgroups.

■ Evaluate Work Processes (How Are We Doing?)

Although evaluating how the project team performed is sometimes done on predictive projects, it isn't a common practice. However, on iterative projects, many companies require an "end of iteration" review of how well the team performed and worked together. One of the advantages of an iterative project is that the same team often stays together for a number of iterations. After each iteration, team members can evaluate how well they worked together and how they can improve their effectiveness and performance as a team. In an Agile project, this is referred to as a **retrospective**. Here are the kinds of questions the team might want to ask:

- Are our communication procedures adequate? How can they be improved?
- Are our working relationships with the user effective?

retrospective a meeting held by the team at the end of an iteration to determine what was successful and what can be improved

- Did we hit our deadlines? Why or why not?
- Did we miss any major issues? How can we avoid this in the future?
- What things went especially well? How can we ensure it continues?
- What were the bottlenecks or problem areas? How can we eliminate them?

■ Monitor Project Progress and Make Corrections

In theory, executing and controlling the project plan sounds easy, but in fact, it is quite complicated. To execute any project, you need some type of project plan. How a team builds and executes project plans will vary depending on whether the project structure is based on a predictive approach or an adaptive approach. In the predictive approach, the project plans are quite large and complex. The adaptive approach is less daunting because the detailed project plan is done for each iteration. Because the piece of work is smaller and often better understood, these plans tend to be smaller and less complex.

Figure 11-19 is a high-level process chart that illustrates the basic process for monitoring and controlling the project. The first box—*Assign work to person or team*—refers to a task that is complex all by itself due to the fact that teams are made up of people with varying skill levels and experiences.

The task for the second box—*Collect status*—is less complex. When collecting status information, you should adhere to the certain guidelines. First, providing status information should be a standard process for all team members. Second, status information should be collected and posted electronically for all to see. Status information can be reported at milestones as complete or not complete.

The task for the third box—*Analyze variance*—requires the project manager to try to determine why the task isn't on target and how significant the delay is with regard to the impact on the total project.

The task for the fourth box—*Take corrective action*—can be complex. Experienced project managers have a whole set of tools they can use to try to correct the variance. Sometimes, the correction is as simple as reassigning team members or maybe it just requires some extra hours of overtime. At other times, tasks may have to be rearranged. In more serious instances, the entire schedule may have to be reworked or more team members may need to be recruited for the team. The objective of corrective action is to get the project back to a known and predictable schedule.

Every development project—whether it follows a predictive or adaptive approach—has lots of questions that need answers and many decisions that need to be made. In many cases, these issues are quickly resolved and the project moves rapidly forward. However, in other instances, the answer to a question or the resolution of an open issue will require additional research. For example,

FIGURE 11-19 Process to monitor and control project execution

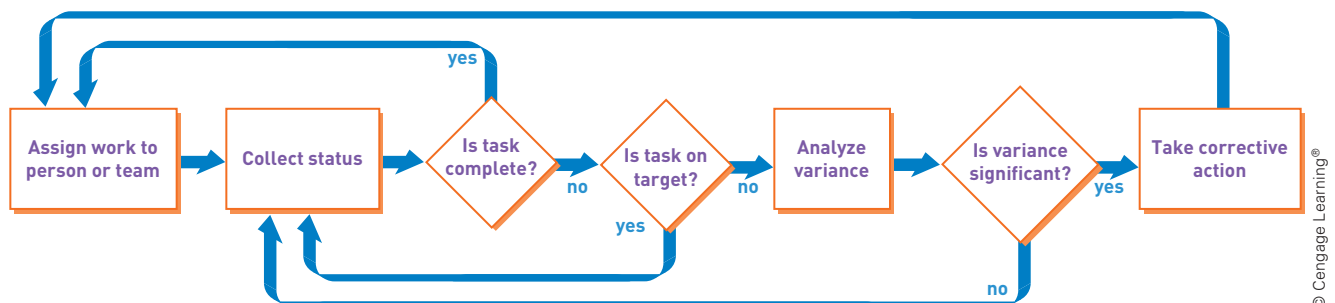


FIGURE 11-20 Sample issue-tracking log

	A	B	C	D	E	F	G	H	I	J	K
1	Issue	Date posted	Short description	Priority	Impact	Reported by	Assigned to	Target date	Resolution description	Status	Fix date
2	101	1/18/2015	Comission structure is undefined	High	Database schema affected	Bill Williams	Ron Workhouse	2/20/2015	Sales VP provided written specifications	Closed	2/1/2014
3	102	2/2/2015	Inconsistent definitions on Customer input form	Urgent	Input form design	Harold Opens	Janice Anay	3/5/2014		Open	
4	103	3/2/2015	New user report identified, but not defined	Medium	New report needs to be generated	Mary Mullins	Mack Dougal	5/5/2014		Open	
5	104										

Source: Microsoft Corporation

a set of rules for sales commissions might include when and how commissions are calculated, what happens to commissions on returned merchandise, when commissions are paid, how the commission schedule varies to encourage sales of high-margin items and sale items, and so on. If management is still making decisions about these rules, you will need to track these issues until they are resolved.

The monitoring and control of open issues and risks for a project is usually no more complex than building various tracking logs. These logs can be built in a simple spreadsheet and posted on the project's Web site or central repository. It is a good idea to make these logs available to all team members. **Figure 11-20** presents an example of a tracking log. The column headings will vary depending on the type of log you use. The tracking log in Figure 11-20 shows issues that need to be resolved by a certain date and the persons responsible for resolving those issues. The log in this figure is a spreadsheet but formatted as a table to allow sorting and filtering by any of the columns.

There are also many issue-tracking systems available that provide additional capabilities such as allowing more detailed descriptions and resolutions. These systems will also provide reports showing open and closed issues and other combinations of issues.

CHAPTER SUMMARY

This chapter focused on the principles and activities related to planning and managing a system development project. It covered three major themes: (1) the principles of project management, (2) the activities to get a project initiated and approved, and (3) the activities to plan the project and monitor its progress.

Project management is the organizing and directing of other people to achieve a planned result. Historically, software projects haven't had a very good track record. Strong project management is seen as one factor that improves success rates of software development projects. Other factors, such as the adaptive approach to the SDLC, can also contribute to project success.

In this chapter's first section, many important skills, techniques, and concepts that relate to project management were discussed. The project management body of knowledge (PMBOK) provides an extensive conceptual foundation for learning about project management. Agile project management also requires the same foundation concepts and skills as the PMBOK,

although many of the specific techniques may be different.

This chapter's second major section focused on the specific activities of Core Process 1, the objective of which is to identify the business need and to get the project initiated. These activities include:

- Identifying the problem
- Extending the project approval factors
- Performing risk and feasibility analysis
- Reviewing with the client and obtaining approval

This chapter's third major section focused on those activities that are necessary to get the project planned, scheduled, and started. These activities include:

- Establishing the project environment
- Scheduling the work
- Staffing and allocating resources
- Evaluating work processes
- Monitoring progress and making corrections

KEY TERMS

break-even point	level of formality or ceremony	retrospective
business benefits	net present value (NPV)	system capabilities
client	oversight committee	System Vision Document
cost/benefit analysis	payback period	tangible benefit
critical path	project iteration schedule	users
detailed work schedule	project management	work breakdown structure (WBS)
Gantt chart	project management body of knowledge (PMBOK)	
intangible benefit		

REVIEW QUESTIONS

- List six major reasons that cause projects to fail.
- Define project management.
- List five internal responsibilities of a project manager.
- List three external responsibilities of a project manager.
- What is the difference between the client and the user?
- What is meant by an organic approach?
- What is the importance of “ceremony”?
- List the 10 areas of the PMBOK.
- What is meant by Agile project management?
- What does the term *chaordic* mean?
- How is scope management accomplished with Agile project management?
- What are the four activities of Core Process 1?
- What are three reasons that projects are initiated?
- What is the difference between system capabilities and business benefits?
- What factors are usually considered when approving a project?
- List 10 types of benefits that may be considered when approving a project.
- Explain how net present value (NPV) is calculated.
- What is the difference between tangible benefits and intangible benefits?
- What are some factors to consider when assessing organizational feasibility?
- List four different types of major risks that may cause problems for a project.
- What are the five activities of Core Process 2?
- What is the purpose of a project dashboard?
- List five types of policies or procedures that must be established for the project team to function.
- What is the difference between the project iteration schedule and the detailed work schedule?
- What is a work breakdown structure used for?
- List three guidelines for identifying tasks on the WBS.
- What is a Gantt chart? What is it used for?
- What is a critical path? Why is it important for the project manager to watch the tasks on the critical path?
- What is the benefit of an iteration review and retrospective?
- Describe in general terms the processes required to monitor project execution.
- What is the usefulness of an issue-tracking system?

PROBLEMS AND EXERCISES

- Read this description and then make a list of expected business benefits that the company might derive from a new system:
Especially for You Jewelers is a small jewelry company in a college town. Over the last couple of years, it has experienced a tremendous increase in its business. However, its financial

performance hasn't kept pace with its growth. The current system, which is partly manual and partly automated, doesn't track accounts receivables sufficiently, and the company is finding it difficult to determine why the receivables are so high. It runs frequent specials to attract customers, but it has no idea whether these are profitable

or if the benefit—if there is one—comes from associated sales. Especially for You wants to increase repeat sales to its existing customers, thus it needs to develop a customer database. It also wants to install a new direct sales and accounting system to help solve these problems.

2. Read this narrative and then make a list of system capabilities for the company:

The new direct sales and accounting system for Especially for You Jewelers will be an important element in the growth and success of the jewelry company. The direct sales portion needs to track every sale and be able to link to the inventory system for cost data to provide a daily profit-and-loss report. The customer database needs to be able to produce purchase histories to assist management in preparing special mailings and special sales to existing customers. Detailed credit balances and aged accounts for each customer would help solve the problem with the high balance of accounts receivables. Special notice letters and credit history reports would help management reduce accounts receivable.

3. Develop a System Vision Document for Especially for You Jewelers based on the work you did for Problem 1 and Problem 2.
4. Develop a work breakdown structure (WBS) based on the following narrative. It should cover all aspects of the move—from the beginning of the project (now) to the end, when all employees are moved into their new offices. Format your solution in tabular form with the following column headings: Task ID No, Task Description, Estimated Effort, and Predecessor Task ID. For your solution, follow these guidelines:

- Include dependencies.
- Include effort (work) estimates.
- Have 30 to 40 detailed tasks.
- Cover a period of at least two months to a maximum of six months.

You are an employee of a small company that has outgrown its facility. It is a Web development and hosting company, so you have technical network administrators, developers, and

a couple people handling marketing and sales. There are 10 employees.

The president of your company has purchased a nearby single-story building, and the company is going to move into it. The building will need some internal modifications to make it suitable. The president has asked you to take charge of the move. Your assignment is to (1) get the building ready, (2) arrange for the move, and (3) carry out the move.

The building is nearly finished, so the job shouldn't be too difficult (no construction is necessary—just some refurbishing). The building has several offices as well as a larger area that needs to be set up with cubicles.

You and the president are walking through the building, and he tells you what he wants:

“Let's use the offices as they are,” she says. “We will need a reception desk for visiting customers. The office in the back corner should be okay for our computer servers. Let's put the salespeople in these offices along the east wall. We are short a few offices, so let's put up a few cubicles in the large room for our junior developers.”

“Of course, we will need to get everybody connected to our system, and I think Ethernet would be faster than wireless for us. And we all need to have phones.”

“Let's plan the move for a long weekend, like a Thursday, Friday, and Saturday. Of course, we need to be careful not to shut down the clients we are already hosting.”

“Will you put together a schedule for the move for our employees and set up instructions for all the employees so they know how they are supposed to get ready for the move? Thanks.”

5. Enter your WBS from Problem 4 into Microsoft Project. First, enter the tasks, dependencies, and durations. Write a paragraph on your experience using Microsoft Project.
6. Develop a six-year NPV spreadsheet similar to the one shown in Figure 11-11. Use the following table of benefits, costs, and discount factors (see **Figure 11-21**). The development costs for the system were \$225,000.

FIGURE 11-21 Benefits, costs, and discount factors for calculating NPV

Year	Annual benefits	Annual operating costs	6% discount factor
1	\$55,000	\$5,000	0.9524
2	\$60,000	\$5,000	0.9070
3	\$70,000	\$5,500	0.8638
4	\$75,000	\$5,500	0.8227
5	\$80,000	\$7,000	0.7835
6	\$80,000	\$8,000	0.7462

FIGURE 11-22 WBS task list for attending a university abroad

Task Id	Description	Duration (days)	Predecessor task
1	Obtain forms from the international exchange office.	1	None
2	Fill out and send in the foreign university application.	3	1
3	Receive approval from the foreign university.	21	2
4	Apply for the scholarship.	3	2
5	Receive notice of approval for the scholarship.	30	4
6	Arrange financing.	5	3, 5
7	Arrange for housing in a dormitory.	25	6
8	Obtain a passport and the required visa.	35	6
9	Send preregistration forms to the university.	2	8
10	Make travel arrangements.	1	7, 9
11	Determine clothing requirements and go shopping.	10	10
12	Pack and make final arrangements to leave.	3	11
13	Travel.	1	12
14	Move into the dormitory.	1	13
15	Finalize registration for classes and other university paperwork.	2	14
16	Begin classes.	1	15

© Cengage Learning®

7. Build a Gantt chart by using Microsoft Project based on the table shown in **Figure 11-22**. Enter the tasks, dependencies, and durations. Print out the PERT chart (Network chart) and the Gantt chart.

Figure 11-22 presents a list of tasks for a student who wants to have an international experience by attending a university abroad. Assume that all predecessor tasks must finish before the succeeding task can begin (the simplest version). Also, insert a few overview tasks, such as Application tasks, Preparation tasks, Travel tasks, and Arrival tasks. Be sure to state your assumption.

8. The state university wants to implement a better system to keep track of all the computer equipment it owns and needs to maintain. The university purchases a tremendous number of computers and software that are distributed throughout the campus and are used by faculty, staff, departments, and colleges. Currently, the university has very sparse records of its equipment and almost no records about maintenance or the software that has been purchased. A list of use cases has been defined; it will serve as the starting point to develop this system.

Take the following list of use cases to create a project iteration schedule. You should try to arrange the use cases so similar ones are developed together. Also, the most important

use cases should be developed first. State your assumptions, and explain your reasons for your solution.

Note: For brevity, we use the word *computer* to refer to any type of computing equipment, such as a desktop computer, laptop computer, server computer, printer, monitor, projector, wireless access point, and so forth.

- Buy a computer.
- Sell a computer.
- Put a computer in service.
- Take a computer out of service (surplus).
- Assign a computer to a person.
- Record the location of a computer.
- Repair a computer (in house).
- Return a computer for repair.
- Identify computers ready for replacement.
- Search for a specific computer by various options.
- Buy a software license.
- Renew a software license.
- Install software on a computer.
- Remove software from a computer.
- Record a warranty for a computer.
- Purchase a warranty for a computer.
- Search for multiple computers by various options.
- Search for software on computers by various options.
- Assign a computer to a department or college.

CASE STUDY

Custom Load Trucking

It was time for Stewart Stockton's annual performance review. As Monica Gibbons, an assistant vice president of information systems, prepared for the interview, she reviewed Stewart's assignments over the last year and his performance. Stewart was one of the "up and coming" systems analysts in the company, and she wanted to be sure to give him solid advice on how to advance his career. For example, she knew that he had a strong desire to become a project manager and accept increasing levels of responsibility. His desire was certainly in agreement with the needs of the company.

Custom Load Trucking (CLT) is a nationwide trucking firm that specializes in the movement of high-tech equipment. With the rapid growth of the communications and computer industries, CLT was feeling more and more pressure from its clients to move its loads more rapidly and precisely. Several new information systems were planned that would enable CLT to schedule and track shipments and truck locations almost to the minute. However, trucking wasn't necessarily a high-interest industry for information systems experts. With the shortage in the job market, CLT had decided not to try to hire project managers for these new projects but to build strong project managers from within the organization.

As Monica reviewed Stewart's record, she found that he had done an excellent job as a team leader on his last project,

where he was a combination team leader and systems analyst on a four-person team. He had been involved in systems analysis, design, and programming, and he had also managed the work of the other three team members. He had assisted in the development of the project schedule and had been able to keep his team right on schedule. It also appeared that the quality of his team's work was as good as, if not better than, other teams on the project. Monica wondered what advice she should give him to help him advance his career. She was also wondering if now was the time to give him his own project.

1. Do you think the decision by CLT to build project managers from its existing employee base is a good one? What advice would you give CLT to make sure it has strong project management skills in the company?
2. What kind of criteria would you develop for Monica to use to measure whether Stewart (or any other potential project manager) is ready for project management responsibility?
3. How would you structure the job for new project managers to ensure or at least increase the possibility of a high level of success?
4. If you were Monica, what kind of advice would you give Stewart about managing his career and attaining his immediate goal of becoming a project manager?

RUNNING CASE STUDIES

Community Board of Realtors®

The Board of Realtors Multiple Listing Service (MLS) system is a fairly focused system. In Chapter 3, you identified a use case diagram for the customer users. In Chapter 8, you extended the functions to include aspects of the system that would be required for the real estate agents to enter their information. You also made some preliminary estimates of iterations and time to complete. Let us expand and refine those answers to include concepts from this chapter.

1. Given the total vision of this system, develop a System Vision Document. Focus primarily on finding the benefits to the community board, the real estate agents, and home buyers.
2. Including the uses cases and functions identified in Chapters 3 and 8, make a list of all the use cases that must be developed. Divide them into subsystems as appropriate. You should have at least two subsystems: one for viewing data and one for updating data. Add any additional use cases (and subsystems) that might be important to the Community Board of Realtors itself. (Hint: Think about user goals and CRUD.)
3. Decide on a work sequence, and develop a project iteration schedule.
4. Estimate the development cost and the time required.
5. Develop a work breakdown structure (WBS) for the project's first iteration.
6. Enter your WBS into Microsoft Project to create a detailed work schedule. (Instructions on how to use Microsoft Project are given in Online Chapter C, which you can find on the Cengage Web site.)

The Spring Breaks ‘R’ Us Travel Service

Assume you are a project manager for Spring Breaks and have been asked to prepare the necessary documents to get this project approved and planned. You have been told that four programmers will be available to work on this project and that it will have the highest priority within the company. In other words, the company would like to have this application up and running as soon as possible. The travel season is fast approaching, and Spring Breaks would like to be able to use the system for this very next season.

As part of the approval and planning activities, you decide that the most important items to develop will be a System Vision Document and a project iteration plan. Given those elements, you can make an estimate of the completion date and the development cost for the system.

1. Based on the answers you gave to the Chapter 2 running case questions, develop a System Vision Document.
2. Based on the functional descriptions you provided for the Chapter 2 running case and the use cases you defined in Chapter 3, finish identifying a complete list of use cases for each of

On the Spot Courier Services

This chapter discussed the first two core processes of a system development project and the activities within these core processes. Obviously, for a normal project, these first two core processes are done at the beginning, when the project manager is still learning about the needs of the business and trying to develop a vision of the solution. However, in this case, you have already gathered a lot of information about On the Spot from the previous chapters. Use this information to develop some project management planning documents.

In Chapter 2, you developed a list of use cases. In Chapter 8, you identified four required subsystems in the total solution. Chapter 6 provided a good review of the essential system capabilities. Using the discussions of On the Spot from these chapters and the items you have produced from your previous work, produce these items:

1. Create a System Vision Document.
2. Review all the use cases that you identified in Chapter 2 and then enhance the list to achieve a complete solution based on the narratives that you have read in previous chapters. In Chapter 5, you learned how to do a CRUD analysis. A CRUD analysis of this case indicates that it might be necessary to add a new subsystem called “Administration.” Assign use cases to

the four subsystems. One important decision you will have to make is which subsystems to develop first. In other words, can the subsystems be deployed independently and, if so, which should be deployed first? Defend your answer.

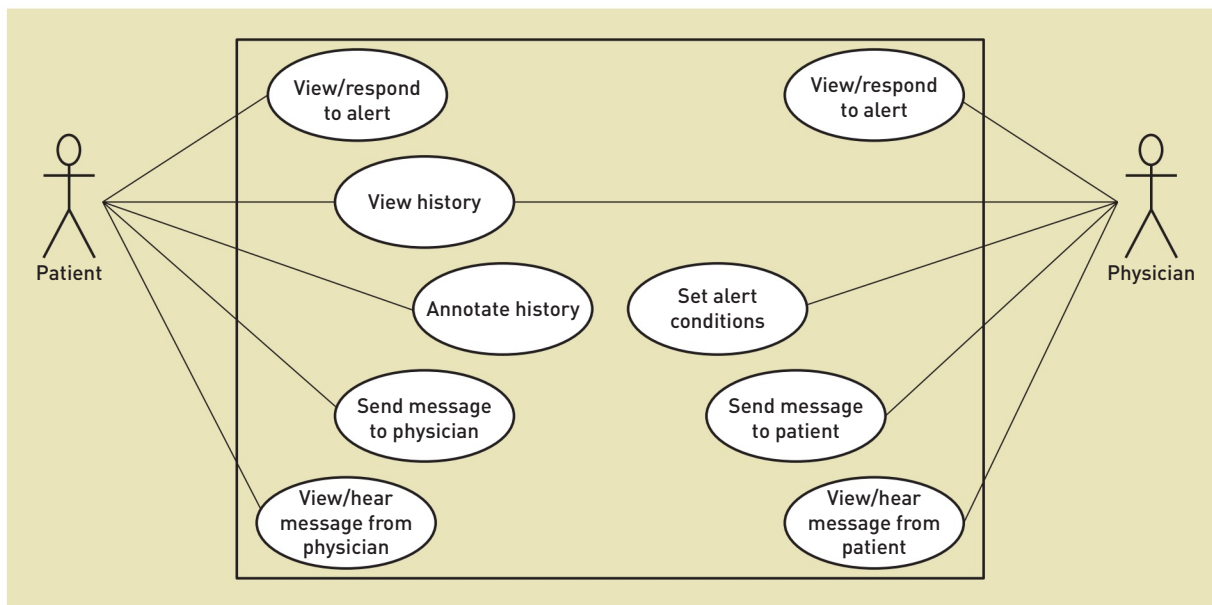
3. A related decision is whether to organize your programmers into one larger team or multiple smaller teams and how many programmers you can use on this project. Make that decision and then defend your answer.
4. Once those decisions are made, develop a project iteration plan. If you have multiple independent teams, your project iteration plan will have parallel paths.
5. Based on your previous answers, develop an estimate for the total project cost and the time required to complete the project.
6. Assuming an annual revenue increase of \$250,000 per year (benefit) and an annual operating cost of \$75,000, develop a five-year NPV worksheet by using your estimates for developing the system. Use a 6 percent discount factor.

the five subsystems; include the Administration subsystem plus the other four subsystems you previously defined:

- Customer account subsystem (like customer account)
- Pickup request subsystem (like sales)
- Package delivery subsystem (like order fulfillment)
- Routing and scheduling subsystem

3. Create a project iteration schedule for each subsystem. The project consultant is planning to assign one team of two people to this project, and the subsystems will be built consecutively. Based on the answers you provided in Chapter 8, combine your four individual schedules into a total project iteration schedule.
4. Create a work breakdown structure (WBS) for the first iteration of the project as you have outlined it. Estimate the effort required for each task in the WBS.
5. Enter the WBS into Microsoft Project to create a detailed work schedule. (Instructions on how to use Microsoft Project are given in Online Chapter C on the Cengage Web site.)

FIGURE 11-23 RTGM system use cases



© Cengage Learning®

Sandia Medical Devices

Use cases were identified for the RTGM system in Chapter 5 (see **Figure 11-23**). Additional descriptions of the system requirements are found in Chapters 3, 4, and 8. You might want to review those to refresh your memory of the needs for this system.

Complete these tasks:

1. Based on the use case diagram and other project information, develop a list of software components (subsystems) that must be acquired or developed. Describe the function(s) of each component in detail. Be sure to consider components that aren't directly tied to use cases, such as the software interface between the glucose monitoring wristband and the cell phone.
2. Prioritize the list of software components based on risk.
3. Prepare a project iteration schedule based on iterations that last between two and four weeks. The schedule should include all the tasks needed to develop a complete version of the system, which will then be subjected to live testing and evaluation by real users for three months.
4. Prepare a detailed work schedule for the first iteration. If you have access to project management software, prepare the schedule and a Gantt chart by using the software.

FURTHER RESOURCES

Scott W. Ambler, *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. John Wiley & Sons, 2002.

Charles G. Cobb, *Making Sense of Agile Project Management: Balancing Control and Agility*. John Wiley & Sons, 2011.

Jim Highsmith, *Agile Project Management: Creating Innovative Products*. John Wiley & Sons, 2009.

Gopal K. Kapur, *Project Management for Information, Technology, Business, and Certification*. Prentice Hall, 2005.

Craig Larman and Bas Vodde, *Scaling Lean and Agile Development: Thinking and Organizational*

Tools for Large-Scale Scrum. Addison-Wesley, 2009.

Jack R. Meredith and Samuel J. Mantel Jr., *Project Management: A Managerial Approach* (6th ed.). John Wiley & Sons, 2004.

Project Management Institute, *A Guide to the Project Management Body of Knowledge* (4th ed.). Project Management Institute, 2008.

Kathy Schwalbe, *Information Technology Project Management* (6th ed.). Course Technology, 2009.

Robert K. Wysocki, *Effective Project Management: Traditional, Agile, Extreme*. John Wiley & Sons, 2009.



Advanced Design and Deployment Concepts

PART FIVE

- **Chapter 12**
Object-Oriented Design:
Fundamentals
- **Chapter 13**
Object-Oriented Design: Use Case
Realization
- **Chapter 14**
Deploying the New System

Object-Oriented Design: Fundamentals

CHAPTER TWELVE

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- ▶ Explain the purpose and objectives of object-oriented design
- ▶ Develop design class diagrams
- ▶ Use CRC cards to define class responsibilities and collaborations
- ▶ Explain important fundamental principles of object-oriented design

CHAPTER OUTLINE

- ▶ Object-Oriented Design: Bridging from Analysis to Implementation
- ▶ Steps of Object-Oriented Design
- ▶ Design Classes and the Design Class Diagram
- ▶ Designing with CRC Cards
- ▶ Fundamental Principles for Good Design

OPENING CASE NEW CAPITAL BANK: PART 1

Bill Santora, the project leader responsible for developing an integrated customer account system at New Capital Bank, just met with the review committee and finished a technical review of the new system's first-cut design. This first-cut design focused on four fundamental core use cases and would be implemented in the first development iteration.

New Capital Bank had been using object-oriented techniques for quite a while, but it had been slower to adopt some of the newer Agile approaches. Bill had been involved in some pilot projects that had used Unified Modeling Language (UML) to develop systems using object-oriented techniques. However, this development project was his first large-scale project that would be entirely Agile.

As Bill was collecting his presentation materials, his supervisor, Mary Garcia, spoke to him: "Your technical review went very well, Bill," she said. "The committee found only a few minor items that need to be fixed. And even though I am not completely current on the new approach, it was easy for me to understand how these core functions will work. I still find it hard to believe that you will have these four pieces implemented in the next few weeks though."

"Wait a minute," Bill said, laughing. "It won't be ready for the users then. Getting these four core functions coded and running doesn't mean that we are almost done. This project will still take a year to complete."

"Yes, I know," Mary said. "But it is nice that we will have something to show after only one month. Not only do I feel more confident in this project, but the users love to see things developing."

"I know," Bill said. "Remember how much grief I got when I originally laid out this plan based on an iterative approach? It was difficult to detail the project schedule for the later iterations, so I had a hard time

convincing everybody that the project schedule wasn't too risky. The upside is that because each iteration is only four weeks long, we have something to show right at the beginning. You don't know how relieved I am that the design passed the review! The team has done a lot of work to make sure the design is solid, and we all felt confident."

"Well, building it incrementally makes a lot of sense and certainly seems to be working," Mary said. "I especially liked the diagrams you showed. It was terrific how the three-layer architectural design supported each use case. Even though I don't consider myself an advanced object-oriented technician, I could understand how the object-oriented design fit into the architecture. I think you wowed everybody when you demonstrated how you could use the same basic design to support both our internal bank tellers and a Web portal for our customers. Congratulations."

Bill picked up on Mary's enthusiasm.

"How about the design class diagrams?" he asked. "Don't they give a nice overview of the classes and the methods? We use them extensively as a focus for discussion on the team. They really help the programmers write good, solid code."

Mary nodded in agreement and added, "By the way, have you scheduled a review with the users?" Mary asked.

"No, not yet," Bill replied. "The architectural design is mostly technical stuff, and we aren't quite ready to meet with the users. The users will help us by verifying our understanding of the information availability, but much of what we do now is too technical for them to follow."

"I am excited to see the first pieces run," Mary said. "It just makes so much sense to be able to test these core functions during the rest of the project. Let me congratulate you again." Then, they headed off to lunch together.

■ Overview

In Chapters 3, 4, and 5, you learned how to do object-oriented analysis by developing functional requirements models. You learned that analysis consists of two parts: discovery and understanding. Discovery is learning exactly how the user conducts his activities through probing questions that provide detailed information. Understanding is taking the information gleaned from user interviews and constructing a set of interrelated and comprehensive models. Model building is an essential part of understanding the user needs and how they influence the proposed system. However, the objective of analysis models isn't to describe the new system, but to understand, in precise terms, the requirements.

In Chapters 6 and 7, you learned about the need and importance of technology architecture and application software architecture. Chapter 7 distinguished the difference between large software components, such as systems or

subsystems, and smaller software components, such as classes and methods. This chapter and the next focus on classes and methods.

This chapter and the next discuss object-oriented design, specifically, on the internal structure of a software system or subsystem. You will learn how to develop object-oriented design models based on the requirements models, which are then used by the programmers to code the system. This chapter first explains design class diagrams, which are an extension of the problem domain class diagram with design information added. Next, it explains class responsibility collaboration (CRC) cards to begin teaching the details of use-case-driven, object-oriented design.

This chapter ends with an important discussion of design principles for good object-oriented design. Throughout this chapter and the next, you learn not only the basics of object-oriented design but also the teaching principles underlying it, so the systems you build are well structured and maintainable. These design principles will provide you with a solid foundation for designing systems correctly.

■ Object-Oriented Design: Bridging from Analysis to Implementation

So, what is object-oriented design? It is a process by which a set of detailed object-oriented design models are built, which are then used by the programmers to write and test programs for the new system. Systems design is the bridge between user requirements and programming the new system. One strength of the object-oriented approach is that the design models are often just extensions of the requirements models. Obviously, it is much easier to extend an existing model than to create entirely new design models. Frequently, developers like to skip the design model step; however, it is a good practice to create design models and not just jump into coding. Just as a builder doesn't build something larger than a doghouse or a shed without a set of blueprints, a good system developer would never try to develop a large system without a set of design models.

One tenet of Agile, adaptive development approaches emphasized throughout this text is to create models only if they have meaning and are necessary. Sometimes, new developers misinterpret this guideline to mean that they don't need to develop design models at all. The design models may not be formalized into a comprehensive set of documents and diagrams, but they are certainly necessary. Developing a system without doing design is comparable to writing a research paper without an outline. You could just sit down and start writing; however, if you want a paper that is cohesive, complete, and comprehensive, you should write an outline first. You could write a complex paper without an outline, but in all probability, it would be disjointed, hard to follow, and missing important points—and it would earn a low grade! The outline can be jotted down on paper, but the process of thinking it through and writing it down allows the writer to ensure that it is cohesive. Systems design provides the same type of framework.

There are three subheadings in this section. First, you'll review how an object-oriented application works. Second, you'll identify the analysis models that are used for object-oriented design and indicate how they flow into the design models. Third, you'll learn about the design models that support the programmers. In other words, the following sections discuss the design models used for programming.

■ Overview of Object-Oriented Programs

Let's quickly review how an object-oriented program works. Many of you will be familiar with these ideas from your programming classes.

instantiation creation of an object based on the template provided by the class definition

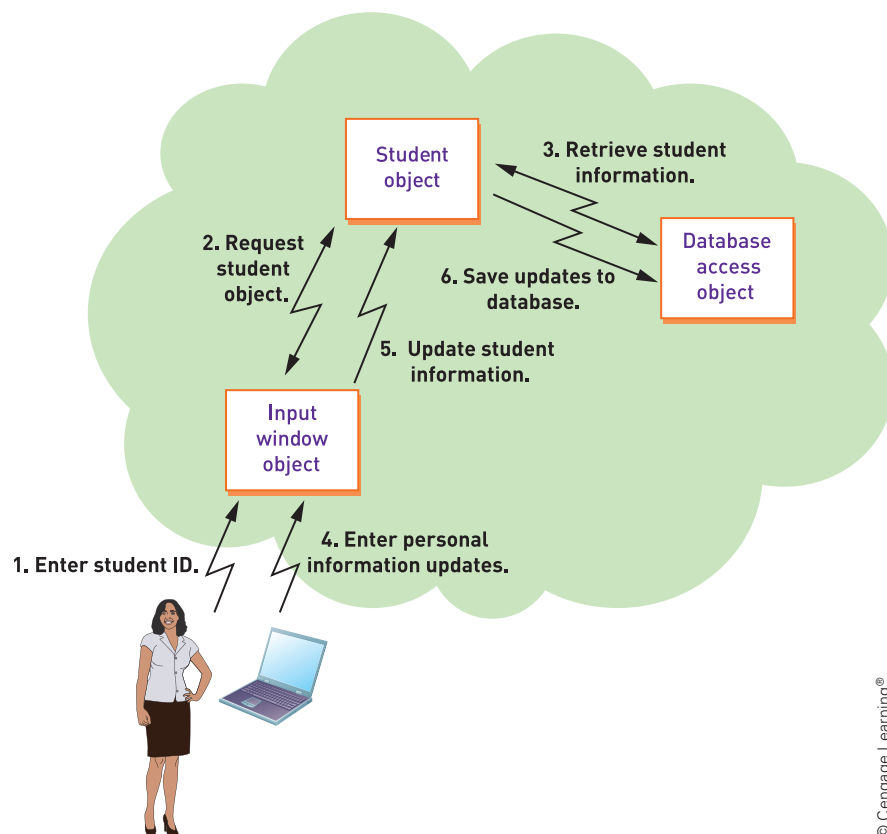
An object-oriented program consists of sets of computing *objects*. Each object has data and program logic encapsulated within itself. Analysts define the structure of the program logic and data fields by defining a *class*. The class definition describes the structure or a template of what an executing object looks like. The object itself doesn't come into existence until the program begins to execute. This is called an **instantiation** of the class—that is, making an instance (an object) based on the template provided by the class definition.

An object-oriented program consists of a set of these instantiated objects that cooperate to accomplish a result. These objects work together by sending each other messages and working in concert to support the functions of the main program.

Figure 12-1 depicts how an object-oriented program works. The program includes a window object that displays a form in which to enter a student ID and other information (message 1). From the user perspective, this object appears as a form on a screen. After the student ID is entered, the Window object sends a message (message 2) to the Student class to tell it to create a new Student object (instance) in the program. The new Student object will know that it needs more data for some of its attributes based on internal logic in the student class, so it will send a message to a Database Access object asking for data from the database (message 3). Once it is completely instantiated with all the required data, the Student object sends the information back to the Window object to display it on the screen. The student user then enters the updates to her personal information (message 4), and another sequence of messages is sent to update the Student object in the program, which forwards information to the Database Access object and writes it to the database.

The important concept you should understand is that an object-oriented program consists of many objects. Within each object, the program logic exists

FIGURE 12-1 Object-oriented event-driven program flow



in small segments called *methods*. These methods are “called” or invoked through messages. In the example in Figure 12-1, there are three objects, each of which has a unique job to do. These three objects can relate back to concepts you learned in Chapter 7 about three-layer architecture. Referring back to Figure 7-11, “Three-layer architecture,” the three-layers translate to the three objects in Figure 12-1: the Window object (view layer), the Student object (domain layer), and the Database Access object (data layer).

You can look inside a class and see the program code that corresponds to the functions performed by the objects once they are instantiated. **Figure 12-2** illustrates an example of the program code for a class with its parameters and methods. Part (a) is the code for Java, and part (b) is the code for VB.NET. The *methods*, such as `getFullName ()`, are the pieces of executable code that carry out the functions of the class.

■ Analysis Models to Design Models

As you remember, during analysis two types of information are captured, which are used to define the requirements: information about things and information about the business processes. A domain model class diagram described the information about things, and use case descriptions, activity diagrams, and system sequence diagrams described the information about the business processes. In other words, requirements are documented as domain classes and business processes.

To write a computer program, a programmer needs to know what the classes are, and what the methods are in those classes. If you think through Figure 12-1 again, note that the example covers a single use case, perhaps one called *Update student information*. This is an important programming concept! A programmer will work on one use case at a time. She will select a use case and program the appropriate methods in the required classes to carry out that use case. Hence, the design models must support that activity. In other words, the design models must provide the information required to (1) identify the classes and (2) document the flow of execution—both for a single use case.

Figure 12-3 illustrates how this model building process flows from analysis to design to implementation. Again, note the two types of information—information about things, and information about the flow or the process. The analysis flow models document the flow of the business process, while the design flow models document the flow of execution through the classes.

■ Introduction to the Design Models

The primary model used to document the classes and the methods is the design class diagram. The design class diagram is an extension of the domain model class diagram that was developed during analysis activities and requirements definition. **Figure 12-4** illustrates a Student class both as the domain class and as the design class. The primary difference is the addition of the method signatures for the class. A method signature includes the method name, the input parameters, and the type for the returned value. Notice that the method names given in the bottom panel of the design class correspond to method names found in the code in Figures 12-2a and b. You’ll learn more about the design class diagram in the section “Design Classes and Design Class Diagram.”

To document the flow of execution of a particular use case, you use the information provided in a UML interaction model or a CRC card. Each of the following three models captures essentially the same information, so you do not need to use all three for any given use case. For a particular use case, you would only use one. The three models are a sequence diagram, a communication diagram, or CRC (class responsibility collaboration) cards. Sequence diagrams and communication diagrams are standard UML interaction diagrams.

FIGURE 12-2a Example of class definition in Java with several methods

```

public class Student
{
    //attributes
    private int studentID;
    private String firstName;
    private String lastName;
    private String street;
    private String city;
    private String state;
    private String zipCode;
    private Date dateAdmitted;
    private float numberCredits;
    private String lastActiveSemester;
    private float lastActiveSemesterGPA;
    private float gradePointAverage;
    private String major;

    //constructors
    public Student (String inFirstName, String inLastName, String inStreet,
        String inCity, String inState, String inZip, Date inDate)
    {
        firstName = inFirstName;
        lastName = inLastName;
        ...
    }
    public Student (int inStudentID)
    {
        //read database to get values
    }

    //get and set methods
    public String getFullName ( )
    {
        return firstName + " " + lastName;
    }
    public void setFirstName (String inFirstName)
    {
        firstName = inFirstName;
    }
    public float getGPA ( )
    {
        return gradePointAverage;
    }
    //and so on

    //processing methods
    public void updateGPA ( )
    {
        //access course records and update lastActiveSemester and
        //to-date credits and GPA
    }
}

```


FIGURE 12-2b Example of class definition in VB.NET with several methods

```

Public Class Student

    'attributes
    Private studentID As Integer
    Private firstName As String
    Private lastName As String
    Private street As String
    Private city As String
    Private state As String
    Private zipCode As String
    Private dateAdmitted As Date
    Private numberCredits As Single
    Private lastActiveSemester As String
    Private lastActiveSemesterGPA As Single
    Private gradePointAverage As Single
    Private major As String

    'constructor methods
    Public Sub New(ByVal inFirstName As String, ByVal inLastName As String,
        ByVal inStreet As String, ByVal inCity As String, ByVal inState As String,
        ByVal inZip As String, ByVal inDate As Date)
        firstName = inFirstName
        lastName = inLastName
        ...
    End Sub

    Public Sub New(ByVal inStudentID)
        'read database to get values
    End Sub

    'get and set accessor methods
    Public Function GetFullName() As String
        Dim info As String
        info = firstName & " " & lastName
        Return info
    End Function

    Public Property firstName()
        Get
            Return firstName
        End Get
        Set (ByVal Value)
            firstName = Value
        End Set
    End Property

    Public ReadOnly Property GPA()
        Get
            Return gradePointAverage
        End Get
    End Property

    'Processing Methods
    Public Function UpdateGPA()
        'read the database and update last semester
        'and to date credits and GPA
    End Function

End Class

```

FIGURE 12-3 Analysis models to design models to programming models

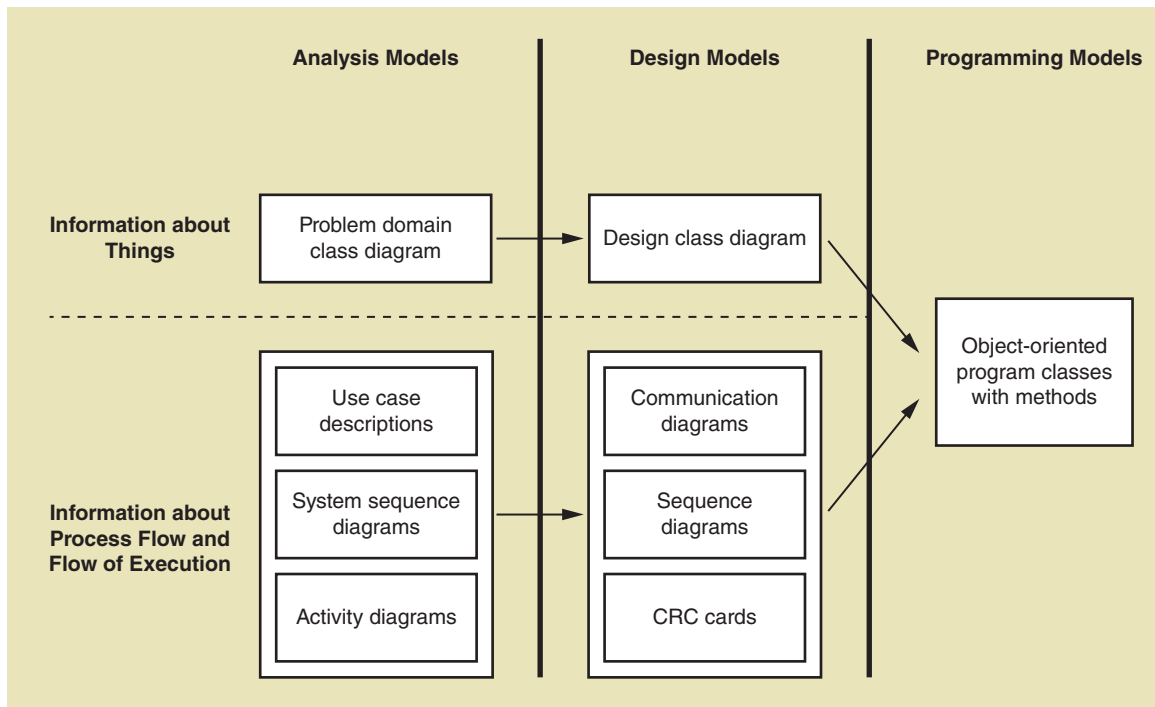


FIGURE 12-4 Student class example with domain class and design class

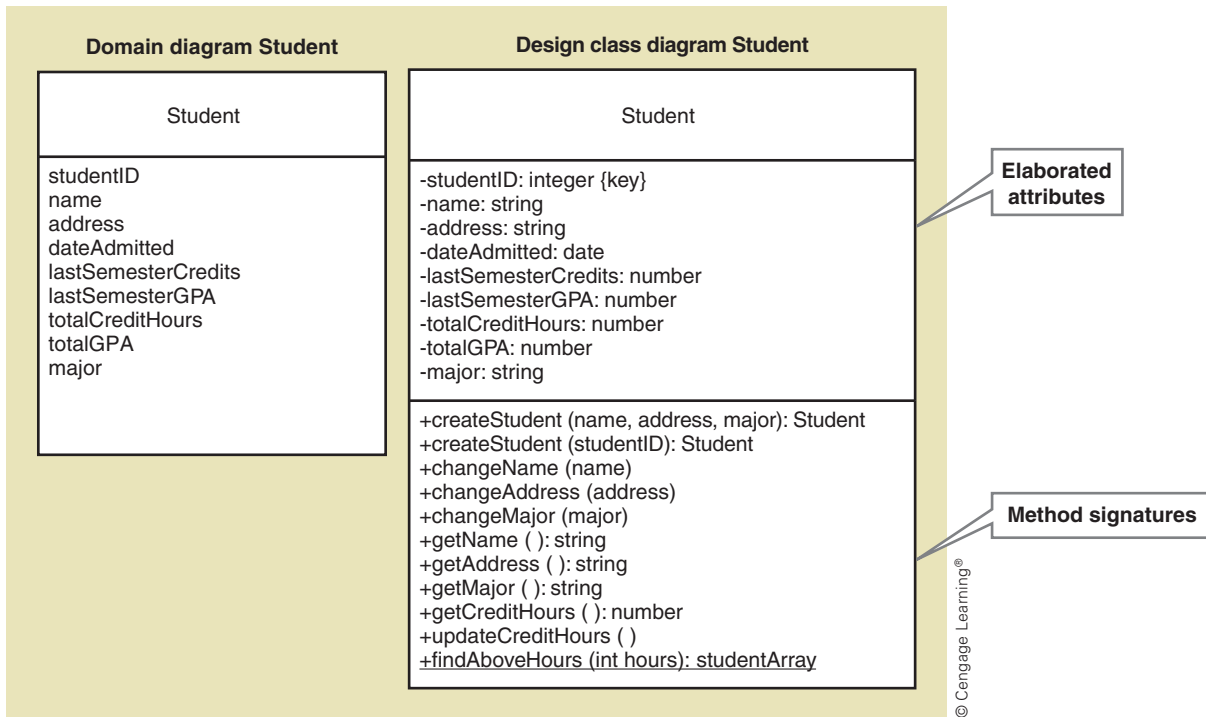
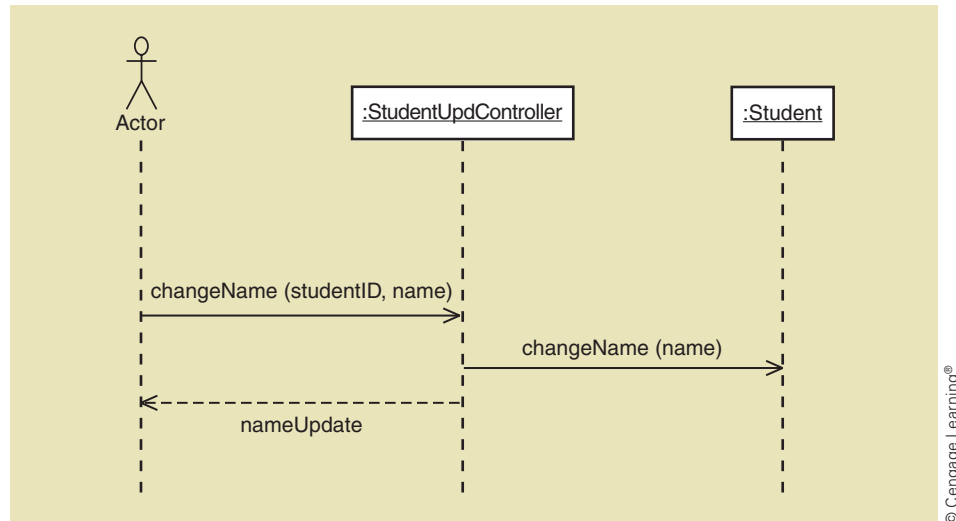


FIGURE 12-5 Sequence diagram for updating student name



CRC cards are not standard UML, but are very popular for designing simple systems. This section introduces you to all three.

You first learned about sequence diagrams in Chapter 5 when you learned about system sequence diagrams (SSD). An SSD only had two actors, the external actor and the system. A sequence diagram expands the system object to identify the internal interacting objects. **Figure 12-5** illustrates a simple sequence diagram. Notice that the System object is no longer included, but two of the internal objects are shown. Also notice the message arrows between the interacting objects.

A communication diagram is similar to a sequence diagram and serves the same purpose. Some developers prefer sequence diagrams, whereas others prefer communication diagrams. Each provides basically the same information, but in different formats. Each also has strengths and weaknesses. **Figure 12-6** illustrates a communication diagram for the same use case as Figure 12-5, updating student name. Chapter 13 elaborates the details and benefits of both sequence and communication diagrams.

The third model that is used to do detailed object-oriented design and to provide information for the design class diagram is CRC cards. Even though the CRC card method is not a standard UML diagramming approach, it provides a straightforward approach for detailed object-oriented design, especially for simple use cases, and thus has become popular. (Remember that object-oriented design is use-case-driven—design is done use case by use case.)

CRC cards include a set of cards, with each card representing a class. Developers use 3 x 5 cards or some other method to document each card. Each card identifies the class, its responsibilities (i.e., its methods), and the other classes with which it collaborates. **Figure 12-7** shows both front and back sides of one single CRC card. The card represents a class. The list on the left is the “responsibilities.” The list on the front right is the other classes with which this class must “collaborate.” In a later section, “Designing with CRC Cards,” you will learn how to use these cards.

FIGURE 12-6 Communication diagram for updating student name

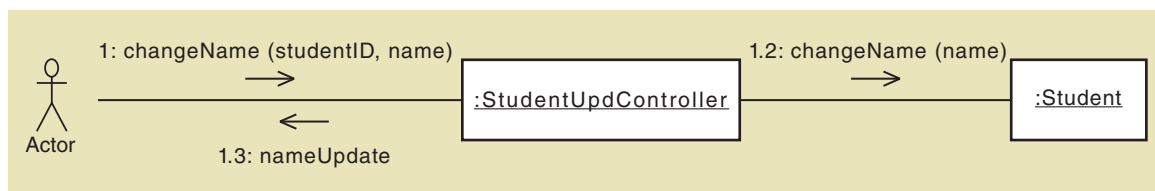
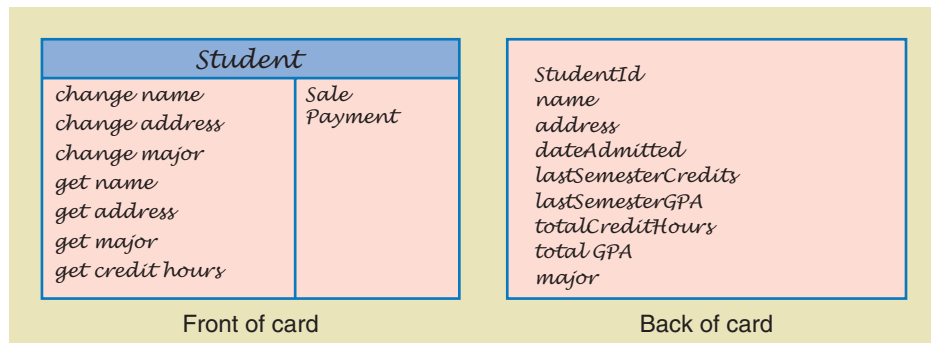


FIGURE 12-7 Sample CRC card for the Student class



An object-oriented system designer must provide enough detail so a programmer can write the initial class definitions, including the method code. The following sections and Chapter 13 explain these models in more depth and explain how to use them to perform detailed design.

■ Steps of Object-Oriented Design

object-oriented design the process to identify the set of classes, their methods, and their messages required to execute a use case

The previous section identified the primary models used for object-oriented design and the source models from analysis activities. As you saw in Figure 12-1, an object-oriented program will have classes in three separate layers: user interface, problem domain, and database access layers. Thus, **object-oriented design** is the process that identifies and describes the classes within each layer and defines the messages that are used to invoke the methods of the involved classes. Object-oriented design is an analytical, rigorous, and detailed process. Don't be discouraged if it takes several tries before you feel comfortable with this skill.

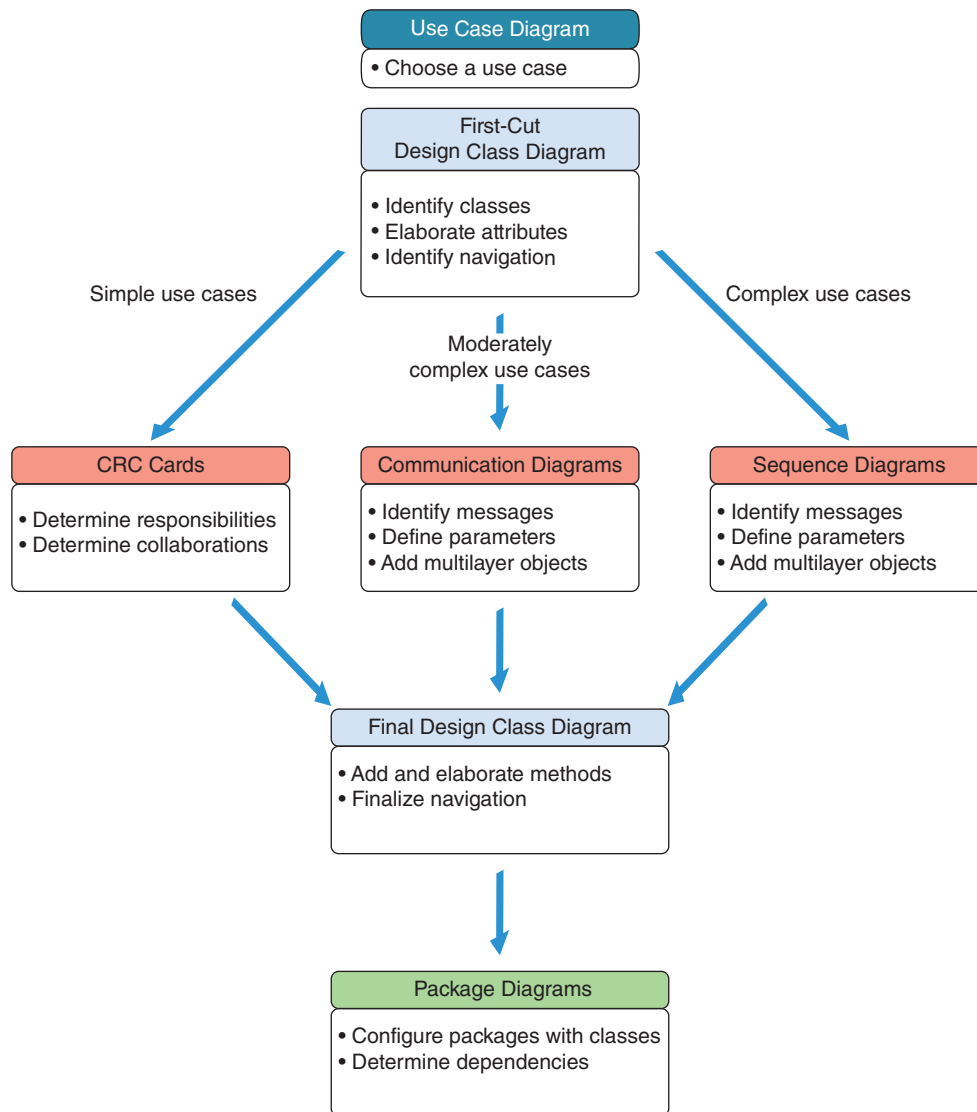
Another important issue about object-oriented design is that it is “use-case-driven.” As was discussed earlier, a programmer will code all of the methods across various classes that are required to implement the flow of execution of a single use case. Object-oriented design is done the same way. A single use case is chosen and the appropriate models are constructed or updated to describe the details of the use case. Note in the definition of object-oriented design, it emphasizes this fact of being use-case-driven.

Because the process of object-oriented design is quite rigorous, sometimes developers are tempted to skip the design step and move directly to programming. In fact, with simple use cases, at times it is easier just to code the use case, rather than develop a formal use case design. As has been discussed earlier with Agile development, the objective of development activities should always be to create accurate, robust software. If a model or diagram will assist in that objective, then it should be created. If it does not contribute directly to the end result, then time should not be spent on it. However, you should not just skip modeling with the assumption that a model is a distraction rather than a necessary step in creating a solid design for the software.

Figure 12-8 illustrates steps required in object-oriented design. The first step is to choose a single use case. Then, you create a model, the type of which depends on the complexity of the use case. The final result is to produce a complete design class diagram with supporting detail as required from interaction diagrams.

Developers will use different types of models depending on the complexity of the use case they are designing. This figure identifies three possible model paths for generating a final design class diagram. The path on the far left, using CRC cards, is often used for use cases that are straightforward and do not

FIGURE 12-8 Object-oriented design process for a single use case



require extensive design decisions. The middle path, using communication diagrams, can be used for use cases that are more complex. The path on the right, using sequence diagrams, can be used for any use case, no matter the level of complexity. No matter which technique is used, detailed design does require a careful thought process to ensure a solid, well-constructed solution.

In any of the three techniques for modeling, the objective is to identify and define the methods that are required in each class. The final design class diagram documents these required methods. It should be noted that each use case will have a separate sequence diagram or communication diagram. Furthermore, the design class diagram is a growing model because as each use case design is finished, the new methods are added to the existing design class diagram. It is a composite model containing the method names for all use cases.

Finally, a package diagram can also be added to divide the classes into components or subsystems that can be implemented as a unit. Some developers use package diagrams, but many do not. The following section begins the detailed discussion about design by going into more depth about the design class diagram.

■ Design Classes and the Design Class Diagram

The design class diagram contains the final definition of each class in the final object-oriented software system. The primary source of information for this diagram is the problem domain model. Hence, the problem domain model serves as the basis for both database design, as you learned in Chapter 9, and for the software classes as defined in the design class diagram. First, you need to understand the diagram itself, and then you'll learn how it is created during the design process.

The domain model class diagram shows a set of problem domain classes and their associations. During analysis, because it is a discovery process, analysts generally don't worry much about the details of the attributes. However, because in object-oriented programming the attributes of a class must be declared as public or private and each attribute must also be defined by its type, such as character or numeric, it is important to elaborate on these details as well as to define the methods and parameters that are passed to the methods and the return values from methods.

As developers build the design class diagrams, they add many more classes that were not originally defined in the domain model. Referring to Figure 12-1, the Input window objects and Database access objects are examples of additional classes that are not problem domain classes. The classes in a system can be partitioned into distinct categories, such as user-interface classes or data access classes. At times, designers may also develop distinct class diagrams by subsystem. The following section now turns to design class diagram notation and discusses the design principles used in developing the first iteration of the design class diagram.

■ Design Class Stereotypes

UML doesn't specifically distinguish between design class notation and domain model class diagram notation. However, practical differences occur simply because the objective of design modeling is distinct from that of domain modeling. Domain modeling shows things in the users' work environment and the naturally occurring associations among them. At that point, the classes aren't specifically software classes. After you start a design class diagram, though, you are specifically defining software classes. Because many different types of design classes are identified during the design process, UML has a special notation—called a *stereotype*—that allows designers to designate a special type of class. A **stereotype** is simply a way to categorize a model element as a certain type. A stereotype extends the basic definition of a model element by indicating that it has some special characteristic you want to highlight. The notation for a stereotype is the name of the type placed within printer's guillemets, like this: «control».

Four types of design classes are considered standard stereotypes: an entity class, a boundary or view class, a controller class, and a data access class. **Figure 12-9** shows the notation used to identify these four stereotypes.

An **entity class** is the design stereotype for a problem domain class. It typically describes something users deal with when doing their work. Objects of entity classes usually need to be remembered and are also referred to as persistent classes. A **persistent class** is an entity stereotyped class with objects that exist after the program quits. However, persistency is usually not indicated as a stereotype. The way to make data persistent is to write it to a file or database so that it is saved and can be retrieved at a later execution of the software.

A **boundary or view class** is specifically designed to live on the system's automation boundary. In a desktop system, these classes would be the windows classes or Web pages and all the other classes associated with the user interface.

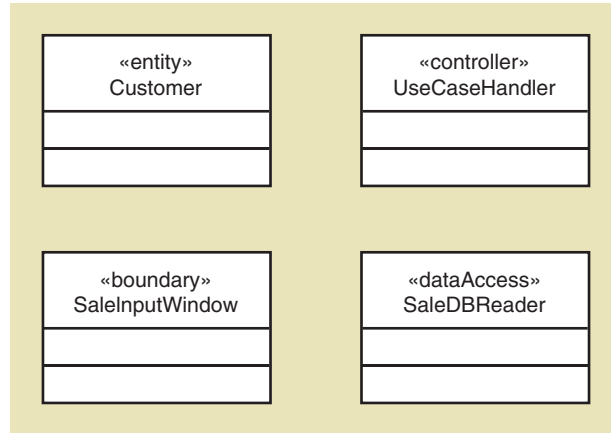
stereotype a way of categorizing a model element by its characteristics, indicated by guillemets (« »)

entity class a design stereotype for a problem domain class

persistent class an entity class whose objects must continue to exist after a system is shut down

boundary or view class a class that exists on a system's automation boundary, such as a user interface or a system interface class

FIGURE 12-9 Standard stereotypes found in UML design models



controller class a class that mediates between boundary classes and entity classes, acting as a switchboard between the boundary or view layer and domain layer

data access class a class that is used to retrieve data from a database

A boundary class could also be a system interface between a company’s system and an external system. So a boundary stereotype is either a user or a system interface class.

A **controller class** mediates between the boundary classes and the entity classes. In other words, its responsibility is to catch the messages from the boundary class objects and send them to the correct entity class objects. It acts as a kind of switchboard between the boundary or view layer and the domain layer.

A **data access class** is used to retrieve data from and send data to a database. Sometimes it is also called a database access class. Rather than insert database access logic, including SQL statements, into the entity class methods, a separate layer of classes to access the database is often included in the design.

■ Design Class Notation

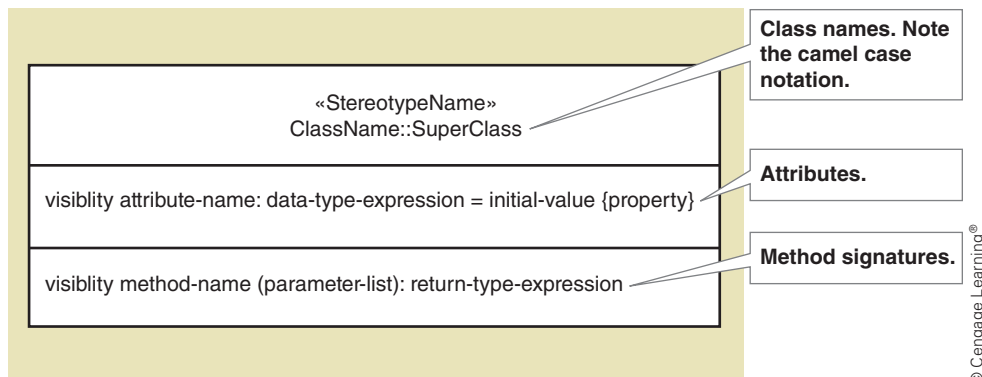
Figure 12-10 shows the details within a design class. Refer back to Figure 12-4 to see an example. The name compartment includes the stereotype name, class name, and superclass name (if any). The lower two compartments contain more details about the attributes and the methods.

The format that analysts use to define each attribute includes the following:

visibility a notation that indicates (by plus or minus sign) whether an attribute can be directly accessed by another object

- Visibility—**Visibility** denotes whether other objects can directly access the attribute. (The values for visibility are a plus sign, which indicates that an attribute is visible, or public, and a minus sign, which indicates that it isn’t visible, or is private, i.e. “+” = visible, “-” = invisible.)
- Attribute-name
- Data-type-expression (such as character, string, integer, number, currency, or date)
- Initial-value, if applicable
- Property (within curly braces), such as {key}, if applicable

FIGURE 12-10 Notation used to define a design class



method signature a notation that shows all the information needed to invoke, or call, the method

The third compartment contains the method signature information. A **method signature** shows all the information needed to invoke (or call) the method. It shows the format of the message that must be sent, which consists of these attributes:

- Method visibility
- Method-name
- Method-parameter-list (incoming arguments)
- Return-type-expression (the type of the return parameter from the method)

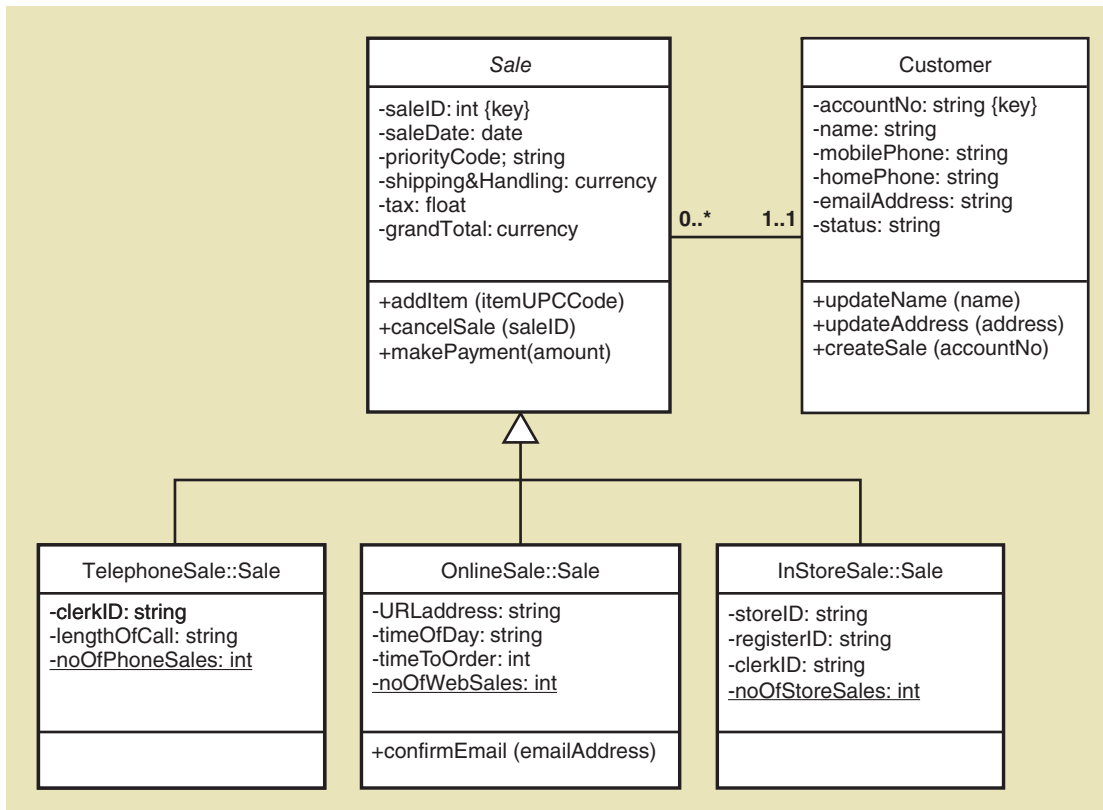
The domain model attribute list contains all attributes discovered during analysis activities. The design class diagram includes more information on attribute types, initial values, and properties. It can also include a stereotype for clarification. As shown in Figure 12-4 in the Student design class diagram, the third compartment contains the method signatures for the class. Remember that UML is meant to be a general object-oriented notation and not specific to any one language. Thus, the notation won't be the same as programming method notation.

The method called findAboveHours (int hours): studentArray, which is denoted with an underline in Figure 12-4, is a special kind of method. Remember that in the object-oriented approach, a class is a template to create individual objects or instances. Most of the methods apply to one instance of the class. However, analysts frequently need a method to look through all the instances at once. Such a method is called a **class-level method** and is denoted by an underline.

class-level method a method that is associated with a class instead of with objects of the class

In Chapter 4, you learned about generalization/specialization. In the problem domain model, generalization/specialization becomes inheritance in the design model and in a programming language. **Figure 12-11** shows an example

FIGURE 12-11 Sale superclass (abstract) with three concrete subclasses showing inheritance



© Cengage Learning®

class-level attribute an attribute that contains the same value for all objects in the system

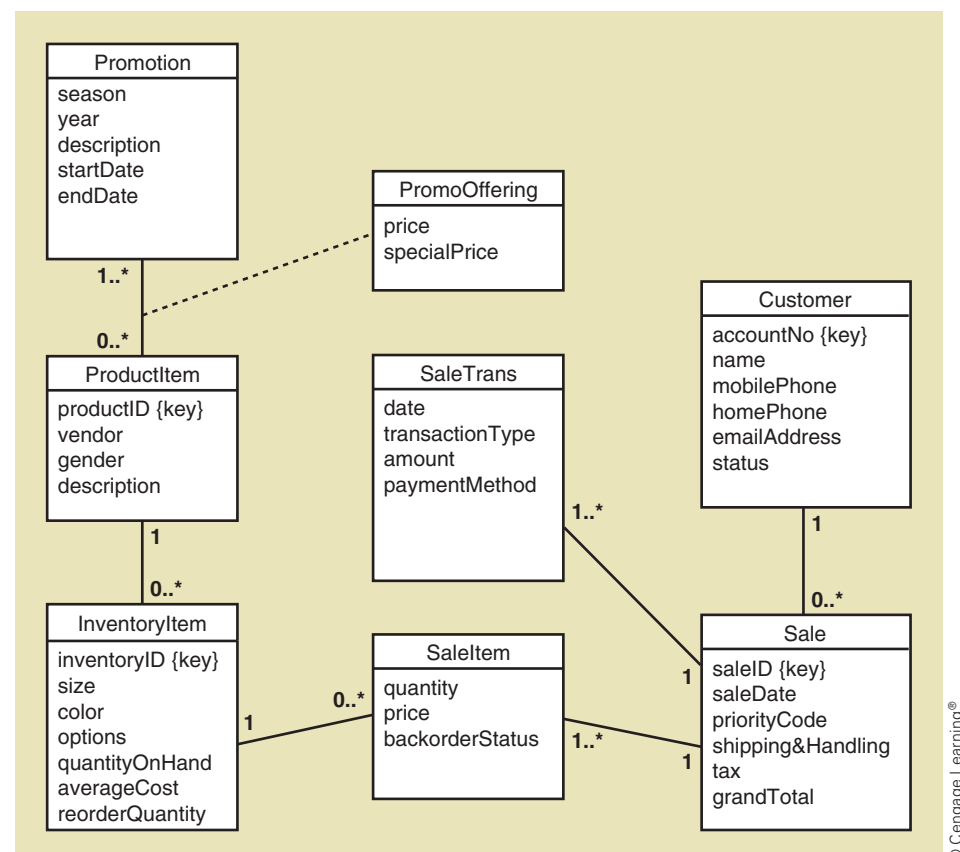
of design classes with attributes and methods; it shows how inheritance works for design classes. Each of the three subclasses inherits all the attributes and methods of the parent Sale class. Hence, each subclass has a saleID, a saleDate, and so forth. In this example, each subclass also has additional attributes that are unique to its own specific class. Each of the subclasses also has a unique attribute that is underlined, such as noOfPhoneSales. As previously mentioned, underlined attributes are **class-level attributes** and have the same characteristics as class-level methods. A class-level attribute is a static variable, and it contains the same value in all instantiated objects of the same type.

Not only are methods and attributes inherited by the subclasses, but associations are also inherited. In Figure 12-11, the Sale object must be associated with exactly one customer. Each subclass inherits the same association; it must be associated with exactly one customer. Finally, notice that the title of the Sale class is italicized. As defined in Chapter 4, an italicized class name indicates that it is an abstract class—a class that can never be instantiated. In other words, there are never any Sale class objects. All orders in the system must be instantiated as one of the three subclasses, which are concrete classes, meaning that every order in the system will be either a TelephoneSale, an OnlineSale, or an InStoreSale. The purpose of an abstract class is illustrated by the figure. It provides a central holding place for all the attributes and methods that each of the three subclasses will need. This example demonstrates one way that OOP implements reuse. The methods and attributes in the abstract superclass only need to be written once in order to be reused by each of the subclasses.

■ Developing the First-Cut Design Class Diagram

To illustrate how to start the design process, this section develops a first-cut design class diagram based on the domain model. **Figure 12-12** is a partial RMO domain model class diagram, as developed in Chapter 4.

FIGURE 12-12 Partial RMO Sales subsystem domain model class diagram



The first-cut design class diagram is developed by extending the domain model class diagram. It requires two steps: (1) adding type and initial value information to the attributes and (2) adding navigation visibility arrows. As indicated earlier, object-oriented design is use-case-driven; so, let's choose a use case to start with and focus only on classes involved in that use case.

■ Elaboration of Attributes

The elaboration of the attributes is fairly straightforward. The type information is determined by the designer, based on his or her expertise. In most instances, all attributes are kept invisible or private and are indicated with minus signs before them. We also need to add a new compartment to each class for the addition of method signatures.

■ Navigation Visibility

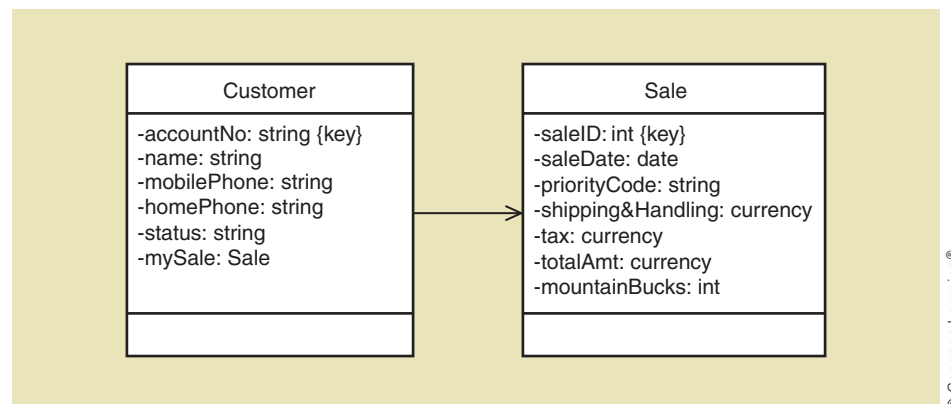
As stated earlier, an object-oriented system is a set of interacting objects. The sequence diagrams document which interactions occur between which objects. However, for one object to interact with another, the first object must be visible to the second object. In this context, **navigation visibility** refers to the ability of one object to interact with and send messages to another object. Sometimes, developers refer to navigation visibility as just *navigation* or *visibility*. However, using just the term visibility can cause confusion with attribute or method visibility. We will use either the term *navigation visibility* or *navigation* to distinguish the concept from public and private visibility on attributes and methods.

Figure 12-13 shows one-way navigation visibility between the Customer class and the Sale class. It is called “one-way” because Customer can interact with Sale, but Sale does not have a direct reference back to Customer. The navigation arrow indicates that a Sale object must be visible to the Customer object. Notice the variable called mySale in the Customer class. This variable refers to a Sale instance. The mySale attribute is included in the example to provide a way to actually implement it. This is quite different from the database concept of including a foreign key. The mySale attribute is not the value of the key to a Sale object, but it is an actual programming reference to the Sale object. You could even think of it as having a Sale object embedded within the Customer object. From a database point of view, you would do exactly the reverse. You would put a foreign key of the Customer in the Sale data record to capture the one-to-many relationship. (One customer has many sales.)

Now let's think about adding navigation visibility to the RMO design class diagram. Remember that you are designing just the first-cut class diagram, so you might continue to modify the navigation arrows as the design progresses.

navigation visibility a design principle in which one object has a reference to another object and thus can interact with it

FIGURE 12-13 Navigation visibility between Customer and Sale



You should ask the following basic question when building navigation visibility: Which classes need to have references to or be able to access which other classes? Here are a few general guidelines:

- One-to-many associations that indicate a superior/subordinate relationship are usually navigated from the superior to the subordinate—for example, from Sale to SaleItem. Sometimes, these relationships form hierarchies of navigation chains—for example, from Promotion to ProductItem to InventoryItem.
- Mandatory associations, where objects in one class can't exist without objects of another class, are usually navigated from the independent class to the dependent class—for example, from Customer to Sale.
- When an object needs information from another object, a navigation arrow might be required, pointing either to the object itself or to its parent in a hierarchy.
- Navigation visibility arrows may be bidirectional—for example, a Sale object might need to send a message to its Customer object as well as the reverse.

The very first step you do when creating the first-cut design class diagram is to add a controller class. You will learn more about controller classes in the next chapter. Basically, the controller class is a switchboard between the input screens and the programming logic classes, i.e. the domain classes, for a particular use case. In our examples, we will always create a controller class for each use case, and place it between the user-interface classes and the problem domain classes.

Figure 12-14 is a first-cut design class diagram for the use case *Create telephone sale* based on the steps described earlier: add a controller class, elaborate attributes, and identify navigation visibility. To elaborate the attributes, you add type information and visibility. To add navigation visibility, you first identify which classes may be involved and then determine the classes that require navigation visibility to other classes. For example, for the *Create telephone sale* use case, price information is in the PromoOffering class and description information is in the ProductItem class. In most instances, it is unnecessary to put the navigation visibility reference attribute in the class. (And, in fact, there are some languages that do not require it.) In other words, the mySale attribute is redundant to the information provided by the arrow. So, even though it was shown in Figure 12-13 to emphasize the concept of navigation visibility, it is left off in Figure 12-14 and subsequent figures.

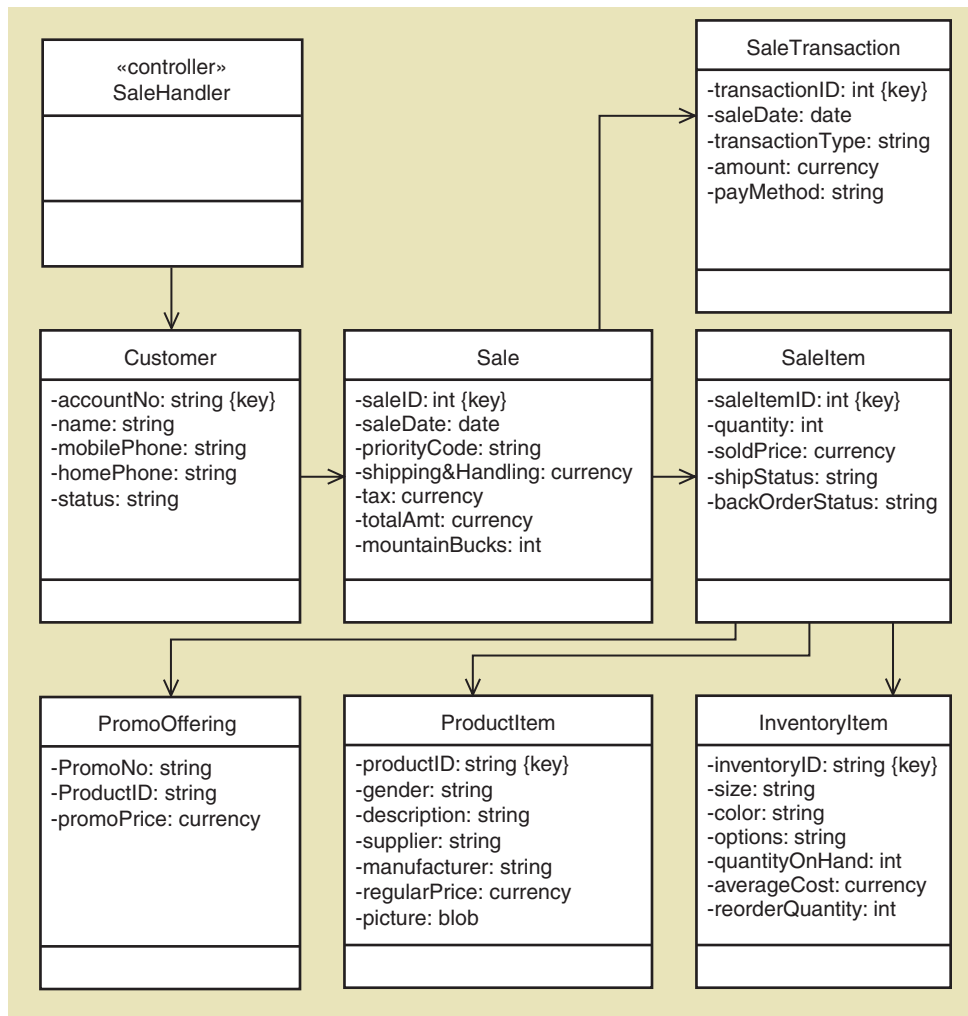
As a reminder, one thing to remember about navigation visibility is that the classes are programming classes, not database classes. So, we aren't thinking about foreign keys in a relational database. We are thinking about object references in a programming language.

Figure 12-14 includes SaleHandler as the controller class. As mentioned previously, a controller class, or use case controller, is a utility class that helps in the processing of a use case. Notice that it has navigation visibility at the top of the visibility hierarchy and starts the use case by messaging a customer.

Three points are important to note: First, as design proceeds use case by use case, you need to ensure that the sequence diagrams support and implement the navigation visibility that was initially defined. Second, the navigation arrows need to be updated as design progresses to be consistent with the design details. Finally, method signatures will be added to each class based on the design decisions made when creating the interaction diagrams for the use cases.

As discussed earlier and shown in Figure 12-8, for use cases that are simple, a developer will often use CRC cards to design the detail flow of execution for the use case from one class to another. The next section explains how CRC cards can be used to design a use case.

FIGURE 12-14 First-cut RMO design class diagram for the Create telephone sale use case



© Cengage Learning®

■ Designing with CRC Cards

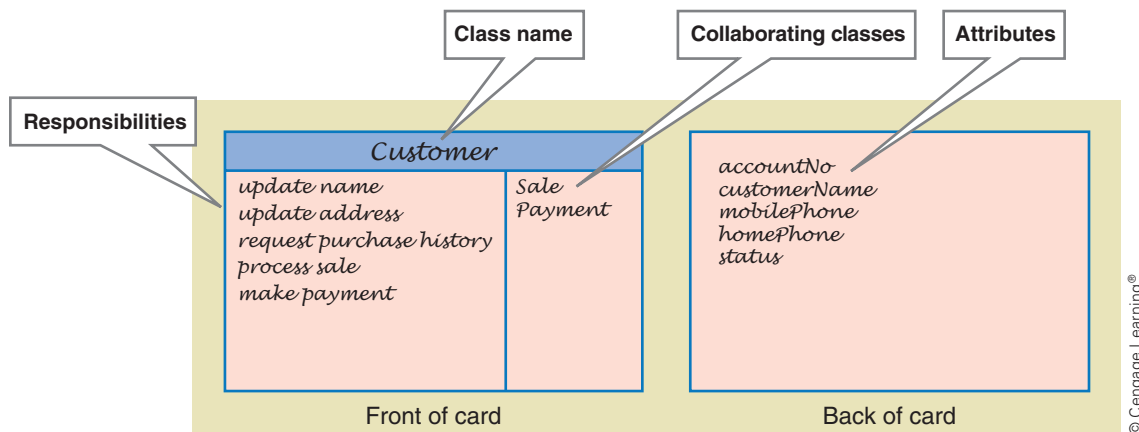
CRC cards a brainstorming and design technique for designing interactions in use cases by assigning responsibilities and collaborations for classes

CRC (class responsibility collaboration) cards are a brainstorming and design technique that is quite popular among object-oriented developers. Developers use this technique during design to help identify responsibilities of the class and the sets of classes that collaborate for a particular use case.

A CRC card is simply a 3×5 or 4×6 index card with lines that partition it into three areas: class name, responsibility, and collaboration classes. **Figure 12-15** illustrates the two sides of a CRC card from the RMO CSMS. The card is partially filled out. Along the top of the card is the name of the class. The left partition lists the responsibilities for objects in this class. Responsibilities include information that the class maintains and actions that the class carries out in support of a particular use case. The right partition lists other classes with which this class collaborates in support of a particular use case. On the back of the card, you have the option of listing important attributes that are required for particular use cases.

The process of developing a CRC model is usually done in a brainstorming session. A design session using CRC cards already has substantial information from which to begin. Before starting the design session, each team member should have a copy of the domain model class diagram. The use case diagram or list of use cases also needs to be available. Other detailed information, such as

FIGURE 12-15 Sample CRC card (front and back)



activity diagrams, system sequence diagrams, and use case descriptions, should be provided, along with a stack of blank CRC-formatted index cards.

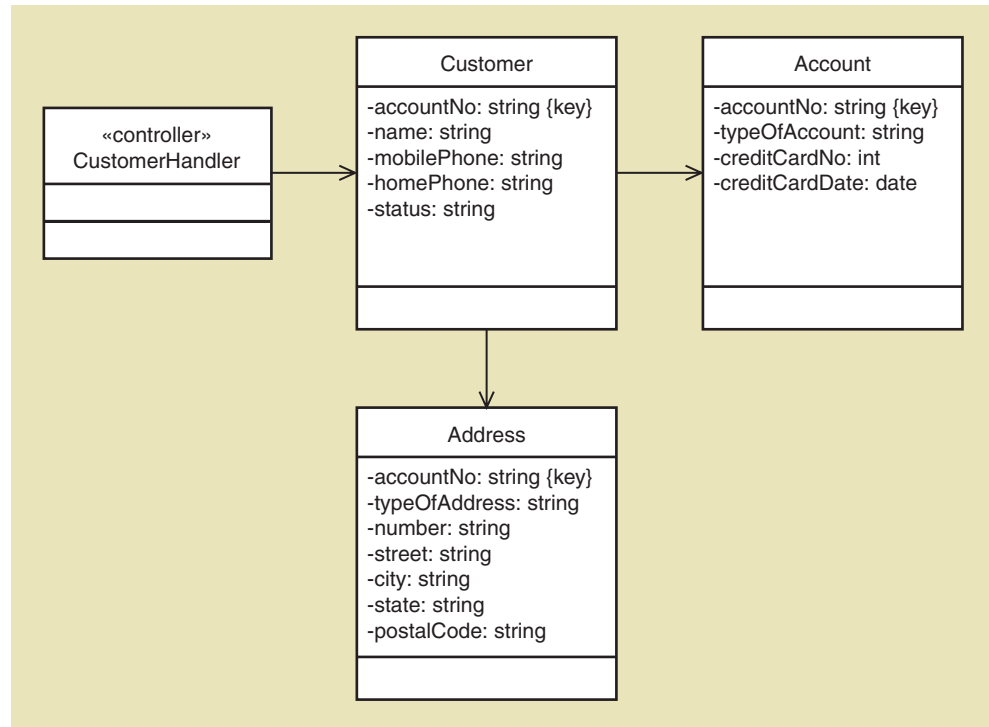
For each use case that needs to be designed, these steps are repeated:

1. Select a use case. Because the process is to design a single use case, start with a set of unused CRC cards. The first card to include should be a use case controller card.
2. Identify the first problem domain class that has responsibility for this use case. This object will receive the first message from the use case controller. Using the domain model that was developed during analysis, select one class to take responsibility. Focus only on the problem domain classes. On the left side of the card, write the object's responsibility. For example, a *Customer* object may take responsibility to make a new sale, so one responsibility may be *Create phone sale*.
3. Identify other classes that must collaborate with the primary class to complete the use case. In other words, identify the classes that have required information or that need to be updated in this use case. As you identify a collaborating class, go to the appropriate CRC card for that class and write its responsibilities on the cards. Also, on the back of each card, write the pertinent information about required attributes of each object class.
4. Another helpful step is to include the user-interface classes. If a user is part of the team and if some preliminary work has been done on the user-interface requirements, it could be effective to add CRC cards for all user-interface window classes that are required for the use case. By including user-interface classes, all the input and output forms can be included in the design, making it much more complete.
5. Add any other required utility classes that are needed to the solution. For example, for a three-layer design, data access classes will be part of the solution. Usually, each persistent domain class will have a data access class to read and write to the database.

At the end of this process, you will have a small set of CRC cards that collaborate to support the use case. This process can be enhanced with several other activities. First, the CRC cards can be arranged on the table in the order they are executed or called. In other words, the calling order can be determined at this time.

To begin, let's work through a simple use case. Referring back to Figure 12-8, we first choose a use case. Next, we create the first-cut design class diagram with navigation. In Chapter 5, we identified a simple use case called

FIGURE 12-16 First-cut design class diagram for Create customer account use case



Create customer account. **Figure 12-16** is the first-cut design class diagram, with navigation visibility added. (Remember that this is simply a preliminary assessment at navigation visibility; it may change as we proceed with the design.) To create this, we first looked at all those domain classes that had association relationships with the Customer class (see Figure 4-24), and selected those that might be created or updated with this use case. For this use case, we only selected the Account class and the Address class.

To learn how the CRC cards process works, let's go through each step listed above and develop the necessary CRC cards. We will use many other requirements models that were created in previous chapters for this use case. In Chapter 5, you saw an activity diagram (refer back to Figure 5-4) and a system sequence diagram (refer back to Figure 5-10), which document the results of analysis activities to determine requirements. That information will also help define the responsibilities of the object classes. So the analysis models identify what needs to be done, but during design we may make changes as we implement how to do it. Remember, however, that during design we may need to make changes based on good design principles and programming techniques.

1. We have selected the use case Create Customer Account. At this point, we will call the controller card, CustomerHandler. **Figure 12-17** illustrates this card. The only responsibility that we have identified so far is a method to create a new Customer object.

FIGURE 12-17 CRC card for CustomerHandler

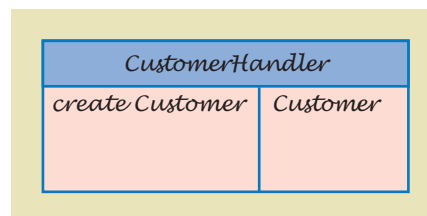
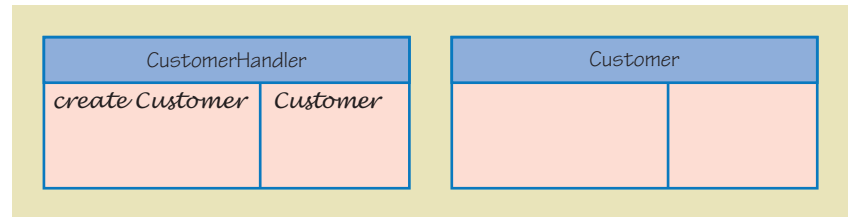


FIGURE 12-18 CRC cards for controller and Customer



2. The primary class responsible for creating a new customer account could be the Customer class, so we will add a CRC card for Customer. It makes sense that a Customer class takes responsibility for creating and updating anything having to do with customer details such as accounts and addresses. The Customer class also has the responsibility to instantiate a new object with a constructor method. Some developers like to include constructor responsibilities, while others do not. In our example, we will not include the constructor. The programmers will know that every class requires at least one constructor. The input information for the constructor can be found in the SSD, as shown in Figure 5-10. **Figure 12-18** illustrates the completion of step 2.
3. For the third step, we observe from Figure 5-4 and Figure 5-10 that there are two other messages coming into the system. One message passes address data and another message passes credit card information. Also note in Figure 12-16 that there are two other classes involved in this use case, Address and Account. When creating a new customer, the system also needs to instantiate a new Account object, and at least one new Address object, though there may also be multiple address objects created. **Figure 12-19** illustrates the completion of step 3, except for the attributes, which are not being shown in this example.
4. For the fourth step, we will add the user-interface classes. Input to this design step will come from two sources, the parameters on the messages as seen in Figure 5-10, and the screen designs and layouts designed during input design. In this instance, let's assume that the user requirement is to have a starting screen that captures basic information such as name, phone numbers, and e-mail addresses. A second input screen will allow entering address information, multiple times if necessary. A final input screen will accept credit card information. Because there may be extra processing required to verify credit information at that point, the user wanted to have it on a separate input screen. **Figure 12-20** illustrates this step in the CRC design process.
5. For the fifth and final step, we add three more CRC cards for the database access classes—one for each of Customer, Address, and Account.

FIGURE 12-19 CRC cards for the problem domain classes

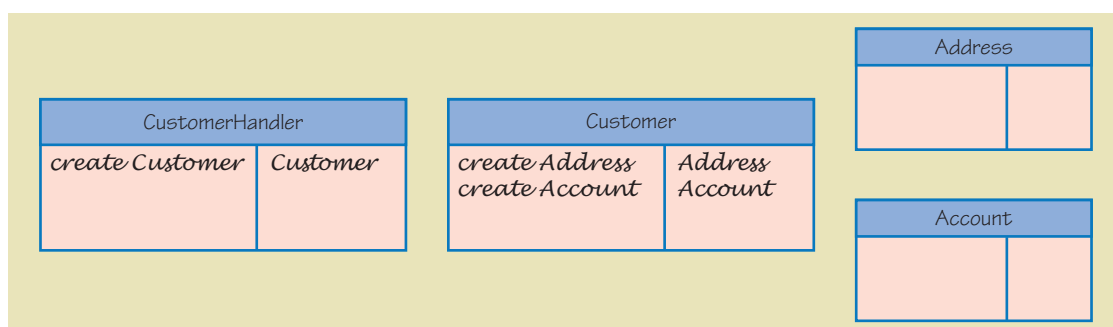
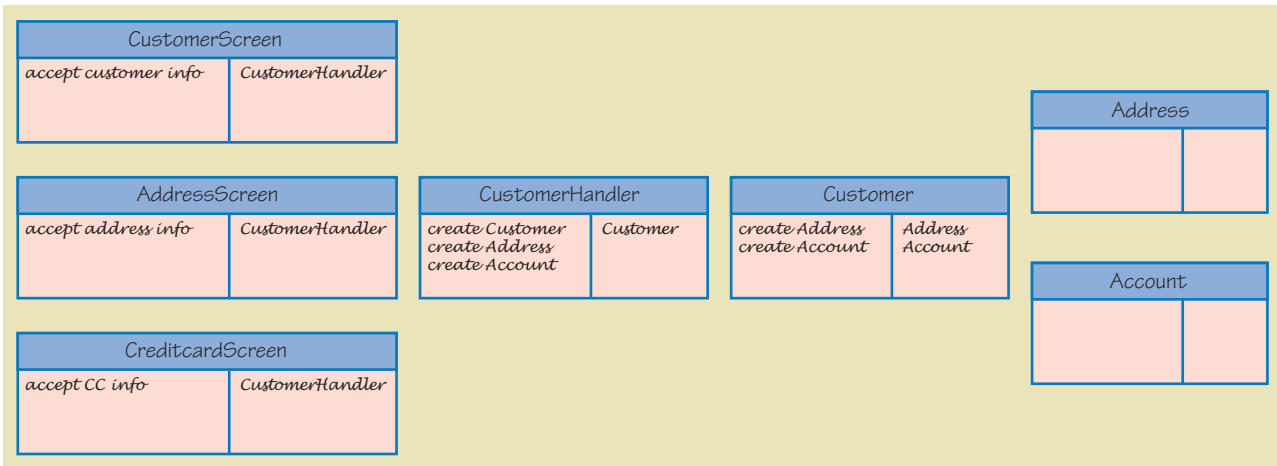


FIGURE 12-20 CRC cards for Create customer account with input classes identified



These three CRC cards are labeled CustomerDB, AddressDB, and AccountDB. We would also need to add collaboration information on the Customer, Address, and Account cards for these three additional classes. Each of these three new cards will have responsibilities to update the appropriate tables in the database. **Figure 12-21** illustrates this final set of CRC cards.

Referring back to our process as described in Figure 12-8, let's update the design class diagram with information from the last set of CRC cards. We initially showed navigation visibility from the controller to Customer, and from Customer to Account and Address. The CRC cards and the decisions we made when creating them confirms our initial assumptions. We could include all the classes, i.e. the user-interface classes and the data access classes, if we so desired; however, for this example we will only include the domain classes and the controller class. To define complete method signatures, we will use the CRC cards

FIGURE 12-21 CRC cards for Create customer account with DB classes

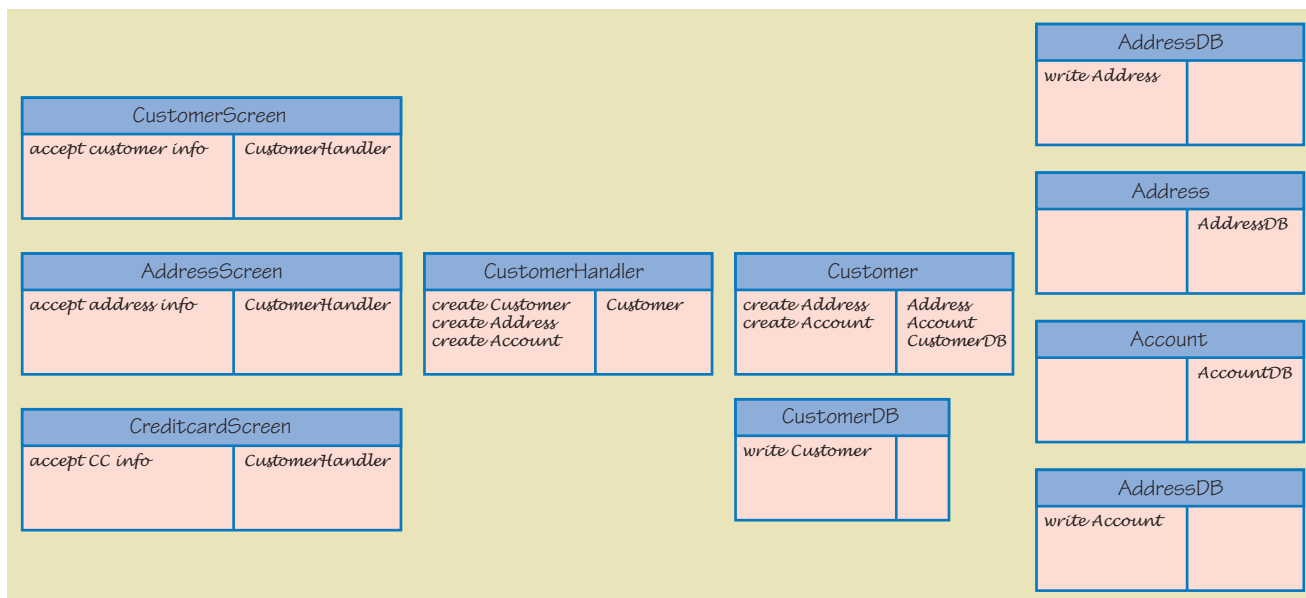
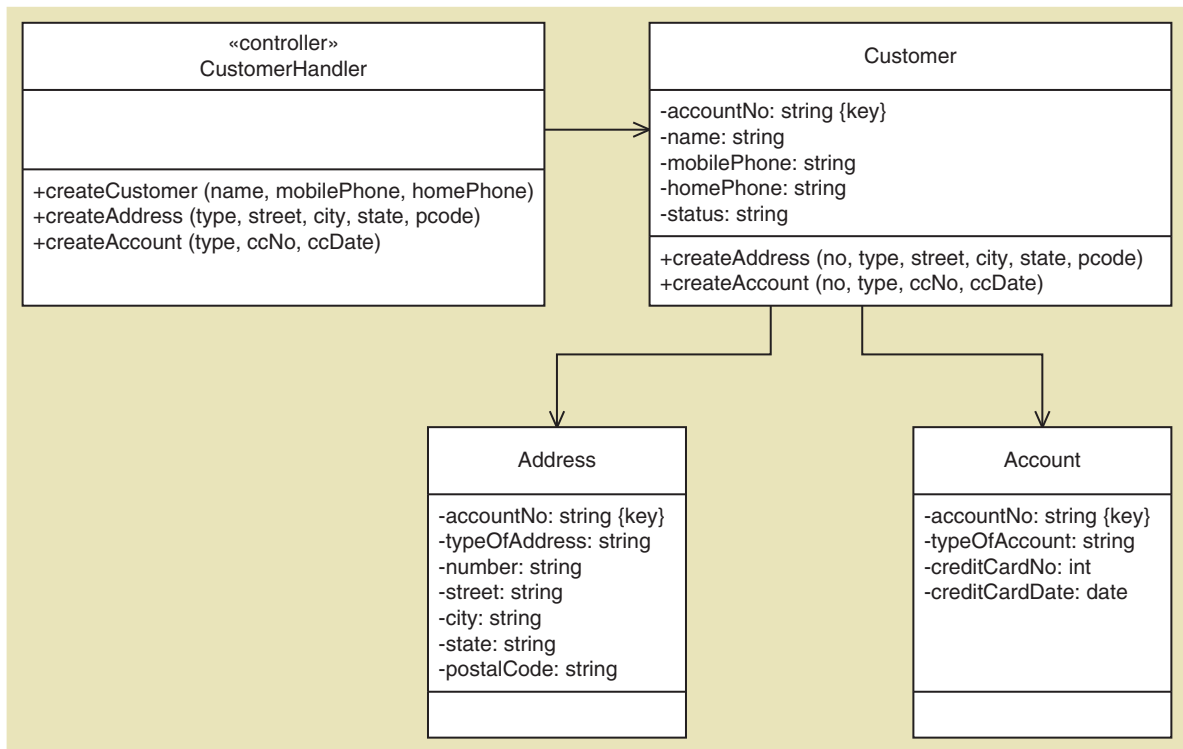


FIGURE 12-22 DCD with method signatures added from CRC cards



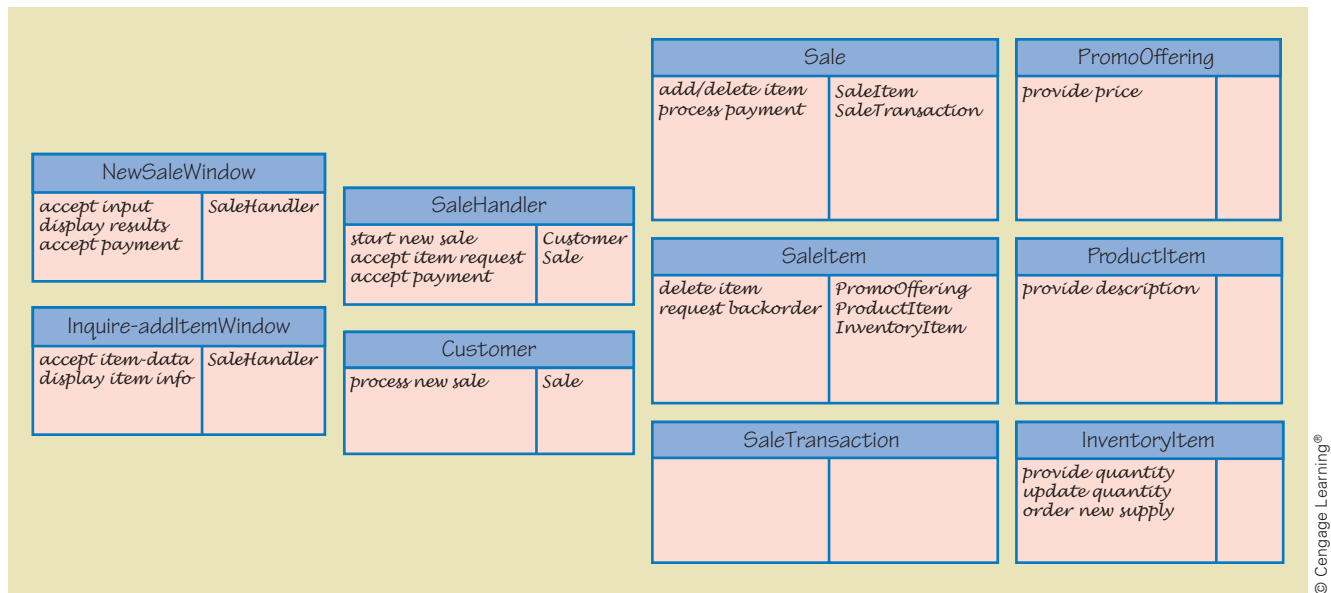
and information from the system sequence diagram (SSD). The methods in the design class diagram are derived from the lists of responsibilities on each card. In other words, each responsibility should translate to a method. The CRC cards identify responsibilities, but do not normally include the complete method signatures with parameters and returns. Not having to worry about parameters simplifies the CRC card process, but it requires that the developer add that information when the design class diagram is completed. Also, because we have elected not to include the constructor responsibilities on CRC cards, we have not included the constructor methods on the final design class diagram. So even though there are no methods defined for *Address* or *Account*, we know that there are constructor methods that execute when new objects are instantiated. If we were to define multiple constructor methods with different parameters and returns, we could add them to this final version of the design class diagram. **Figure 12-22** shows the updated design class diagram of the domain layer with method signatures added for this use case.

Let's show one more example. Figure 12-14, which we developed to illustrate design class diagram elements, contains the first-cut design class diagram for the RMO use case *Create telephone sale*. We will complete this example and show the set of CRC cards and the final design class diagram.

For this example, it makes sense that the *Customer* object creates a *Sale* object, the *Sale* object creates *SaleItem* objects, and *SaleItem* objects access *ProductItem*, *InventoryItem*, and *PromoOffering* objects to get required information. We had determined this when we created the first-cut design class diagram in Figure 12-14. **Figure 12-23** illustrates a solution set of CRC cards for the use case *Create telephone sale*. The *process payment* responsibility in the *Sale* class will cause a *SaleTransaction* to be created.

To finish the example, let us go back to the design class diagram and update it based on the design information created during the CRC card brainstorming

FIGURE 12-23 CRC cards model for Create telephone sale use case



© Cengage Learning®

session. **Figure 12-24** shows an updated design class diagram, with methods navigation visibility updates added. Note that the SaleHandler class needs visibility to the Sale class to process a payment. Compare the responsibilities identified on the CRC cards and the method names described in each class. Note the close correlation. Also remember that we have not included constructors. So for example, the SaleTransaction has a constructor method, which is not shown, to process the payment by instantiating a new SaleTransaction object.

Often, when developers begin using CRC cards, they assign many different responsibilities for a given class. For example, developers might say the SaleItem class should get the price. In reality, the PromoOffering class provides the price and the SaleItem only uses it. In other words, it helps to think of responsibilities as being similar to methods—requests to do something rather than random actions that need to occur.

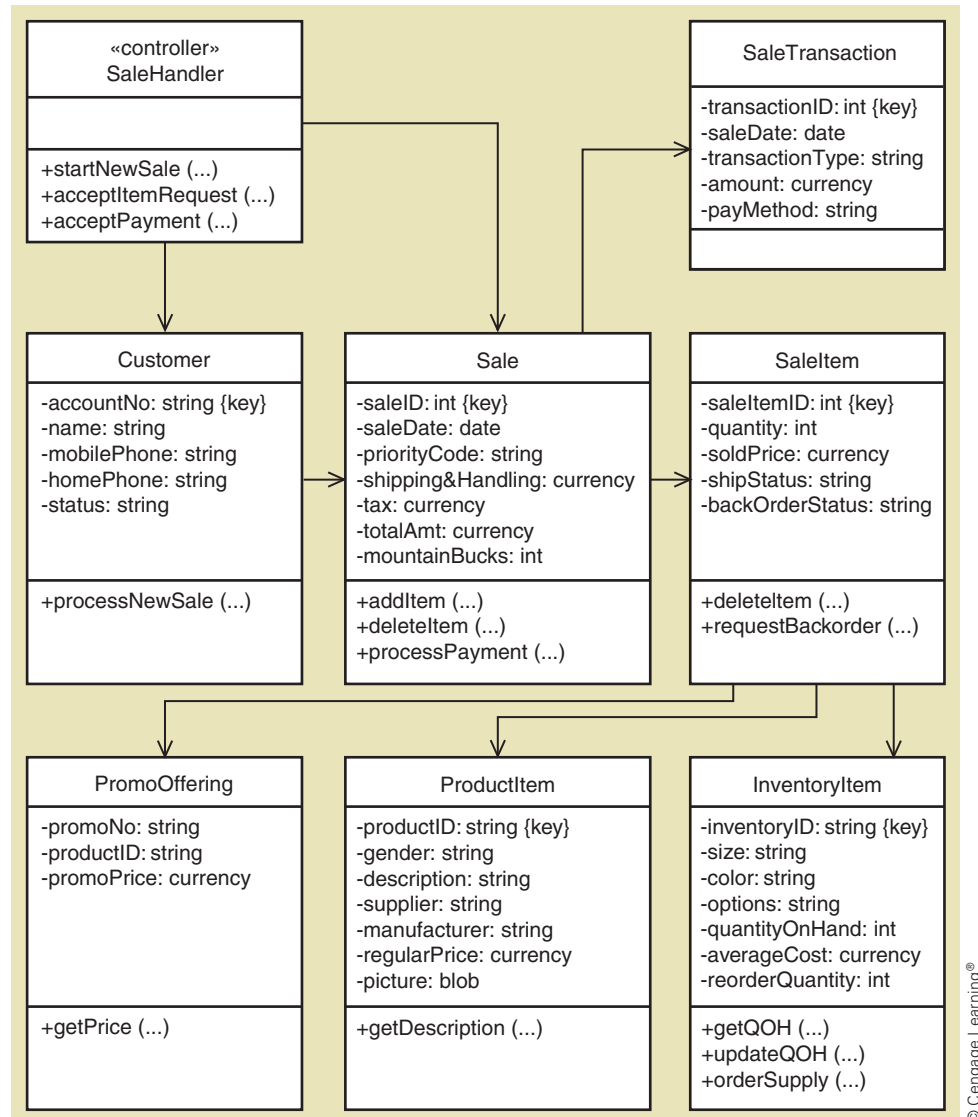
As method names and navigation visibility are added to the design class diagram for each use case, the overall diagram grows and becomes the central repository of all information about every class in the new system. Each set of CRC cards, or interaction diagrams as shown in the next chapter, can be used for programming the new system. The overall design class diagram becomes a source for verification that all use cases have been programmed completely and accurately.

Now that you have some idea of the design process, let's go back through and think about what makes a good design.

■ Fundamental Principles for Good Design

With something as complex as a software application, there are probably an infinite number of design and programming solutions that will work. Some solutions will be “good” solutions, while many other solutions would be considered “bad” solutions. In Online Chapter C, “Project Management Techniques,” the PMBOK area of Project Quality Management identified several characteristics of “good” software. Included were several characteristics related to usability of the software application and fitness for its intended use. Other characteristics focused on robustness, reliability, and maintainability of the software itself.

FIGURE 12-24 Updated DCD for Create telephone sale use case



© Cengage Learning®

These characteristics determine the quality of the design. In other words, they indicate the “goodness” of the detailed design and the implementation of the new system. Although understanding and using the following design principles will not guarantee a high-quality design, applying these principles will increase the probability that the design is solid. Ignoring the following principles will almost certainly ensure a less-than-optimal software design.

The following discussion of good design principles is a launching point for you to understand how to design high-quality software. However, it is not a comprehensive discussion. There are many good books and articles on the principles of good object-oriented design, which further explain principles and techniques of good design.

■ Object Responsibility

One of the most fundamental principles of object-oriented design is the idea of **object responsibility**; that is, objects should be responsible for carrying out system processing. These responsibilities are categorized in two major areas: knowing and doing. In other words, what is an object expected to know and what is an object expected to do or to initiate?

object responsibility a design principle in which objects are responsible for carrying out system processing

Knowing includes an object's responsibilities for knowing about its own data and knowing about other classes with which it must collaborate to carry out use cases. Obviously, a class should know about its own data, what attributes exist, and how to maintain the information in those attributes. It should also know where to go to get information when required. For example, during the instantiation of an object, data that aren't passed as parameters may be required. An object should know about or have navigation visibility for other objects that can provide the required information. For example, in Figure 12-2, the first constructor method for the Student class doesn't receive a studentID value as a parameter. Instead, the Student class takes responsibility for creating a new studentID. It will know how to create a studentID according to an internal algorithm, such as an "autonumber next higher number" algorithm, or perhaps a globally unique identifier routine.

Doing includes all the activities an object performs to complete a use case. Some of those activities include receiving and processing messages. Another activity is to instantiate, or create, new objects that may be required for completion of a use case. Classes must collaborate to carry out a use case, and some classes are responsible for coordinating the collaboration. For example, in the use case *Create phone sale*, the Sale class has responsibility to create SaleItem objects. Another class, such as InventoryItem, is only responsible for providing information about itself. Good design dictates that the developer pay close attention to recognizing and assigning responsibilities to the appropriate objects.

■ Separation of Responsibilities or Separation of Concerns

separation of responsibilities a design principle that recommends segregating classes into separate packages or groupings based on a primary focus of processing responsibility

Separation of responsibilities, also called separation of concerns, is a design principle that is applied to a group of classes rather than to each class individually. The basic idea of separation of responsibilities is to segregate classes into packages or groupings based on a primary focus of processing responsibility. Separation of responsibilities is the fundamental principle behind multilayer design. In multilayer design, there are user-interface classes, business logic classes, and data access classes. Each layer has a particular focus or area of responsibility. Classes that share the same focus or concern are grouped together in a layer. This design principle allows flexibility in system deployment because different layers, i.e., a grouping of classes, can be located on different computers or at different locations.

■ Protection from Variations

protection from variations a design principle in which parts of a system that are unlikely to change are segregated from those that will

Another underlying principle of good design is **protection from variations**—the principle that the parts of a system that are unlikely to change should be segregated (or protected) from those that will change. As you design systems, you should try to isolate the parts that will change from those that are more stable.

Protection from variations is a principle that affects the multilayer design pattern. Designers could mix all the user-interface logic and business logic together in the same classes. In fact, in early user-oriented, event-driven systems, such as those built with early versions of Visual Basic and PowerBuilder, the business logic was included in the view layer classes—often in the Windows input forms. Many Web applications also combine HTML and business logic. The problem with this design was that when an interface needed to be updated, all the business logic had to be rewritten. A better approach is to decouple the user-interface logic from the business logic. Then, the user interface can be rewritten without affecting the business logic. In other words, the business logic—being more stable—is protected from variations in the user interface.

Also, what if updates to the business logic require the addition of new classes and new methods? For example, if the user interface is included as part of the business classes, then changing the business logic would impact the user

interface. However, building the system with a use case controller allows the user interface to simply send all its input messages to the use case controller class. Thus, changes to the methods or classes in the business logic and domain layer are isolated to the controller class. You will find that protection from variations affects almost every design decision, so you should watch for and recognize the application of this principle in all design activities.

■ Indirection

indirection a design principle in which an intermediate class is placed between two classes to decouple them but still link them

Indirection is the principle of separating two classes or other system components by placing an intermediate class between them to serve as a link. In other words, instructions don't go directly from A to B; they are sent through C first. Or in message terminology, don't send a message from A to B. Let A send the message to C and then let C forward it to B.

Although there are many ways to implement protection from variations, indirection is frequently used for that purpose. Inserting an intermediate object allows any variation in one system to be isolated in that intermediate object. Indirection is also useful for many corporate security systems. For example, many companies have firewalls and proxy servers that receive and send messages between an internal network and the Internet. A proxy server appears as a real server—ready to receive such messages as e-mail and HTML page requests. However, it is a fake server, which catches all the messages and redistributes them to the recipients. This indirection step allows security controls to be put in place and protect the system.

A common example of indirection that we have been using throughout is the insertion of a use case controller. The controller is a separate class that receives all the inputs and directs it to the appropriate domain classes.

■ Coupling

This principle, coupling, and the next principle, cohesion, were originally defined during the early days of software design and programming. However, both principles continue to be extremely important and applicable to object-oriented design.

coupling a qualitative measure of how closely the classes in a design class diagram are linked

In the previous examples throughout the text of the RMO class diagram, you have seen that the Customer class and the Sale class are linked together in an association relationship. Another term for this linking is coupled. **Coupling** is a qualitative measure of how closely the classes in a design class diagram are linked. A simple way to think about coupling is by the number of association relationships and whole/part relationships on the design class diagram. Previously, you learned about navigation visibility, which measures what a class can link to and access. Low coupling is usually better for a system than high coupling. For example, a Customer object can access a Sale object that belongs to it. However, if it can also directly access the SaleItem, that would be too much coupling. Only the Sale object should be able to access its own SaleItem objects.

We say that coupling is a qualitative measure because no specific number measures coupling in a system. It is a global measure that applies to a set of classes for a particular use case design. It does not measure an individual class. A designer must develop a feel for coupling—that is, recognize when there is too much or too little. Coupling is evaluated as a design progresses—use case by use case. Generally, if each use case design has a reasonable level of coupling, the entire system will too.

Refer back to Figure 12-1 to observe the flow of messages between the objects. Obviously, objects that send messages to each other must have visibility and thus are coupled. For the Input window object to send a message to the Student object, it must have visibility, or be coupled, to it. The Input window object isn't connected to the Database access object, so those objects aren't coupled.

If we designed the system so the Input window object accessed the Database access object, the overall coupling for this use case would increase; that is, there would be more connections. Is that good or bad? In this simple example, it might not be a problem. But for a system with 10 or more classes, too many connections with visibility can cause high levels of coupling, making the system more complex and therefore harder to maintain.

So, why is high coupling bad? The main reason is that a change in one class ripples through the entire system and may cause methods to fail. Therefore, experienced analysts make every effort to simplify coupling and reduce ripple effects in the design of a new system.

■ Cohesion

cohesion a qualitative measure of the focus or unity of purpose within a single class

Cohesion refers to the consistency of the functions within a single class and is a qualitative measure of its focus or unity of purpose. Unlike coupling (where you want low coupling), classes need to be highly cohesive to be well designed. For example, in Figure 12-1, you would expect the Student class to have methods—that is, functions—to enter student information, such as identification number or name. That would represent a unity of purpose and a highly cohesive class. But what if that same object also had methods to make classroom assignments or assign professors to courses? The cohesiveness of the class would be reduced.

Classes with low cohesion have several negative effects. First, they are hard to maintain. Because they perform many different functions, they tend to be overly sensitive to changes within the system, suffering from ripple effects. Second, it is hard to reuse such classes because they have many different—and often unrelated—functions. For example, a button class that processes button clicks can easily be reused. However, a button class that processes both button clicks and user log-ins has limited reusability. A final drawback is that classes with low cohesion are usually difficult to understand. Frequently, their functions are intertwined and their logic is complex.

Although there is no firm metric to measure cohesiveness, we can think about classes as having very low, low, medium, or high cohesion. Remember, high cohesion is the most desirable. An example of very low cohesion is a class that has responsibility for services in different functional areas, such as a class that accesses both the Internet and a database. These two types of activities are different and accomplish different purposes. To put them together in one class causes very low cohesion.

An example of low cohesion is a class that has different responsibilities, but in related functional areas—for example, a class that handles all database access for every table in the database. It would be better to have different classes to access customer information, order information, and inventory information. Although the functions are the same—that is, they access the database—the types of data passed and retrieved are very different. Thus, a class that is connected to the entire database isn't as reusable as one that is only connected to the Customer table.

An example of medium cohesion is a class that has closely related responsibilities, such as a single class that maintains customer information and customer account information. A better design would be to define two highly cohesive classes: One class could be defined for customer information, such as names and addresses, and another class or set of classes could be defined for customer accounts, such as balances, payments, credit information, and all financial activity. If the customer information and the account information are limited, they could be combined into a single class with medium cohesiveness. Either medium or highly cohesive classes can be acceptable in systems design.

Now let's begin the detailed design process by investigating the properties and details of the design class diagram.

CHAPTER SUMMARY

The ultimate responsibility of system developers is to write computer software that solves a business problem. This chapter focuses on how to configure and develop the solution system—that is, how to design the details of the new system. Systems design is the bridge that puts business requirements in terms that the programmers can use to write the software that becomes the solution system.

Using all the requirements models as well as the architectural design, object-oriented design extends the models so programming can proceed. The objective of object-oriented design is to determine the methods within individual classes that are needed to implement the use cases. The process of design is use-case-driven, in that it is done one use case at a time.

The process of object-oriented design can be divided into two major areas: developing a design class diagram (DCD) and identifying the methods for each use case via an interaction diagram. The DCD is usually developed in two steps. A first-cut DCD is created based on the domain model class diagram, but then it

is refined and expanded as the sequence diagrams are developed. One method of determining which objects collaborate is to use class responsibility collaboration (CRC) cards. For simple use cases, a set of CRC cards may be sufficient to write code. For more complex use cases, other interaction diagrams are normally used.

One reason that we suggest a more formal system of design, rather than just starting to write code is that the final system is much more robust and maintainable. Design as a rigorous activity builds better systems. Some fundamental principles should be considered as a system is developed; specifically, two critical ideas are coupling and cohesion. A good system has low coupling between the classes, and each of the classes has high cohesion. Another important principle is “protection from variations,” meaning that some parts of the system should be protected from and not tightly coupled to other parts of the system that are less stable and subject to change. Being a good developer entails learning and following the principles of good design.

KEY TERMS

boundary or view class
class-level attribute
class-level method
cohesion
controller class
coupling
CRC (class responsibility
collaboration) cards

data access class
entity class
indirection
instantiation
method signature
navigation visibility
object-oriented design

object responsibility
persistent class
protection from variations
separation of responsibilities
stereotype
visibility

REVIEW QUESTIONS

- Describe in your own words how an object-oriented program works.
- What is instantiation?
- List the models that are used for object-oriented systems design.
- Explain how domain classes are different from design classes.
- What is the difference between a system sequence diagram and a sequence diagram?
- In your own words, list the flow of steps for doing object-oriented design.
- What do we mean by use-case-driven design?
- Explain in your own words what coupling means and why it is important.
- Explain what cohesion means and why it is important.
- Compare and contrast the ideas of coupling and cohesion.
- What is protection from variations, and why is it important in detailed design?
- What is meant by object responsibility, and why is it important in detailed design?

13. What is meant by separation of responsibilities?
14. What are (a) persistent classes, (b) entity classes, (c) boundary classes, (d) controller classes, and (e) data access classes?
15. What are class-level methods and class-level attributes?
16. What is a method signature?
17. Compare and contrast abstract and concrete classes. Give an example of each.
18. What information is added to the domain model to derive the first-cut DCD?
19. Describe navigation visibility. Why is it important in detailed design?
20. List some typical conditions that dictate in which direction navigation visibility occurs.
21. What information is maintained on CRC cards?
22. What is the objective of a CRC card design session?
23. How are CRC cards used to update the DCD?

PROBLEMS AND EXERCISES

1. In this chapter, we developed a first-cut DCD, a set of CRC cards, and a final DCD for the *Create customer account* use case for RMO. Create the same three drawings for the *Ship items* use case. (Hint: Figure 5-3 contains a use case description.)
2. Find a company that does object-oriented design using CRC cards. The information systems unit at your university often uses object-oriented techniques. Sit in on a CRC design brainstorming session. Interview some of the developers about their feelings regarding the effectiveness of doing CRC design. Find out what documentation remains after the sessions and how it is used.
3. Find a system that was developed by using Java. If possible, find one that has an Internet user interface and a network-based user interface. Is it multilayered—three layers or two layers? Can you identify the view layer classes, the domain layer classes, and the data access layer classes?
4. Find a system that was developed by using Visual Studio .NET (Visual Basic or C#). If possible, find one that has an Internet user interface and a network-based user interface. Is it multilayered? Where is the business logic? Can you identify the view layer classes, the domain layer classes, and the data access layer classes?
5. Pick an OOP language with which you are familiar. Find a programming IDE tool that supports that language. Test its reverse-engineering capabilities to generate UML class diagrams from existing code. Evaluate how well it does and how easy the models are to use. Can it input UML diagrams and generate skeletal class definitions? Write a report on how it works and what UML models it can generate.
6. Draw a UML design class that shows the following information:
The class name is Boat, and it is a concrete entity class. All three attributes are private strings with initial null values. The attribute *boat identifier* has the property of “key.” The other attributes are the manufacturer of the boat and the model of the boat. There is also an integer class-level attribute containing the total count of all boat objects that have been instantiated. Boat methods include creating a new instance; updating the manufacturer; updating the model; and getting the boat identifier, manufacturer, and model year. There is a class-level method for getting the count of all boats.

CASE STUDY

The State Patrol Ticket-Processing System (Revisited)

In Chapter 3, you identified use cases and considered the domain classes for the State Patrol ticket-processing system. Review the descriptions in that chapter for the use case *Record a traffic ticket*. Recall that the domain classes included Driver, Officer, Ticket, and Court.

1. Draw a DCD for the ticket-processing system based on the four classes just listed and include attributes, association, and multiplicity.
2. List the classes that would be involved in the use cases and decide which class should be responsible for collaborating with the other classes for the use case *Record a traffic ticket*. Consider some possibilities: (1) A Driver object should be responsible for

recording his/her ticket, (2) the Officer object should be responsible for recording the ticket that he or she writes, and (3) a Ticket object should be responsible for recording itself.

3. Create a set of CRC cards showing these classes, responsibilities, and collaborations for the use case.
4. Draw a DCD based on your CRC cards. Include method names.

RUNNING CASE STUDIES

Community Board of Realtors®

In Chapter 3 and Chapter 5, you identified and then modeled use cases for the Multiple Listing Service (MLS) application. You also identified and modeled domain classes. Use your solutions from these chapters to do the following:

1. Draw a first-cut design class diagram (DCD) based on the domain classes for the *Create new listing* use case.
2. Use the CRC cards technique to verify the classes that are involved in the *Create new*

listing use case. Recall that creating a new listing involves an agent, a real estate office, and a listing. Decide which class should have the primary responsibility for collaborating with the other classes and then complete the CRC cards for the use case.

3. Update the design class diagram with method names from the CRC cards. For this solution, do not try to determine entire method signatures, just the names.

The Spring Breaks 'R' Us Travel Service

In Chapter 3, you identified use cases for the Spring Breaks 'R' Us Travel Service. In Chapter 4, you identified the classes associated with these use cases. In Chapter 5, you elaborated those use cases. Using your solutions from these chapters, do the following:

1. Draw a first-cut design class diagram (DCD) for the *Book a reservation* use case.
2. Use the CRC cards technique to verify the classes that are involved in the *Book a*

reservation use case. Recall that creating a booking involves at least a student group, a resort, a week, and a room type. Decide which class should have the primary responsibility for collaborating with the other classes and then complete the CRC cards for the use case.

3. Update the design class diagram with method names from the CRC cards. For this solution, do not try to determine entire method signatures, just the names.

On the Spot Courier Services

In Chapter 6, you considered the issues relevant to the specification of the hardware equipment and networking requirements. The case description in Chapter 6 also reviewed the three primary types of users for the system and many of their respective system-supported activities.

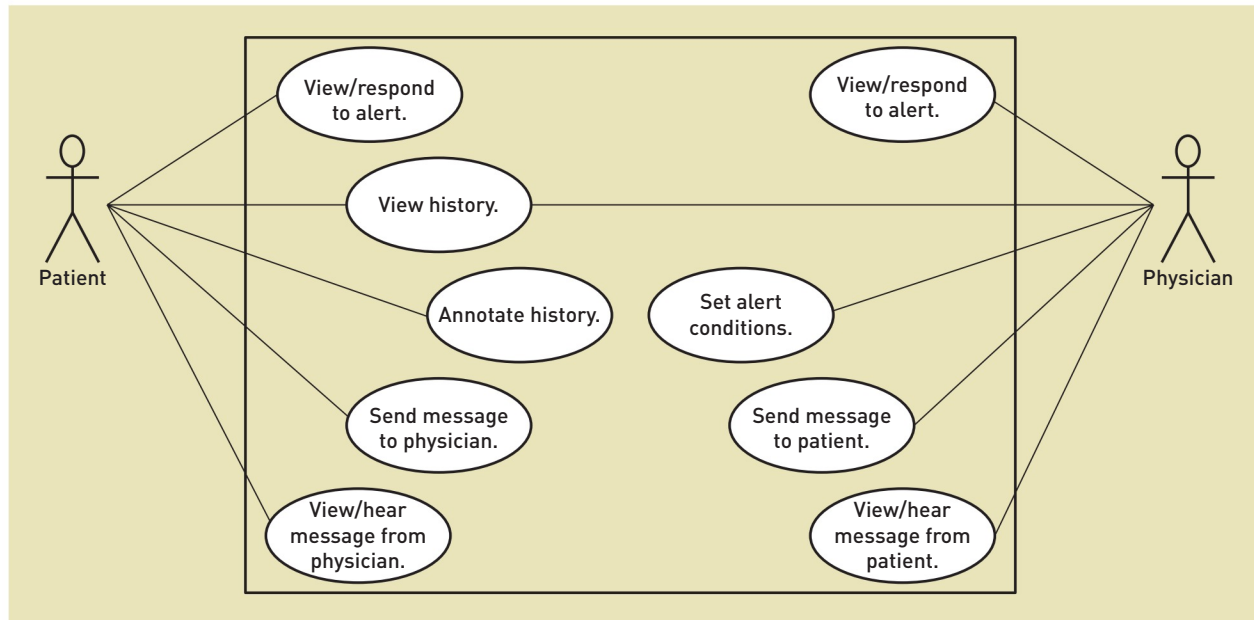
In Chapter 5, you developed activity diagrams and system sequence diagrams for two use cases: *Request a package pickup* and *Pickup a package*. In Chapter 4, you developed a domain model class diagram for the system.

1. For each of the two use cases, develop a first-cut DCD and a set of CRC cards. The design class diagram should elaborate

the attributes and show navigation visibility. You may also need to add more classes from your solution in Chapter 4. It isn't uncommon for developers to enhance early models as they begin to understand system requirements better. The CRC cards should include classes for the controller class and any classes for screens you identified in Chapter 7.

2. Update the DCD with method names from the CRC cards. For this solution, do not try to determine entire method signatures, just the names.

FIGURE 12-25 Use cases for the patient and physician actors



Sandia Medical Devices

Review the original system description in Chapter 2, additional project information in Chapters 3, 4, 6, 8, and 9, and the use case diagram shown in Figure 12-25 to refamiliarize yourself with the proposed system.

Complete these tasks:

1. For the moment, assume that the database will store two glucose levels for each patient—normal minimum and normal maximum—and that an alert will be generated if three or more consecutive glucose readings are above or below those levels. Expand the domain class diagram in Chapter 4 to include this information and then develop a first-cut design class diagram to support the patient use case *View/respond to alert*.
2. Develop a design solution using CRC cards for the *View/respond to alert* use case.

FURTHER RESOURCES

- Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.
- Grady Booch, et al., *Object-Oriented Analysis and Design with Applications* (3rd ed.). Addison-Wesley, 2007.
- E. Reed Doke, J. W. Satzinger, and S. R. Williams, *Object-Oriented Application Development Using Java*. Course Technology, 2002.
- E. Reed Doke, J. W. Satzinger, and S. R. Williams, *Object-Oriented Application Development Using Microsoft Visual Basic .NET*. Course Technology, 2003.
- Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado, *UML 2 Toolkit*. John Wiley & Sons, 2004.
- Martin Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (3rd ed.). Addison-Wesley, 2004.
- Ivar Jacobson, Grady Booch, and James Rumbaugh, *The Unified Software Development Process*. Addison-Wesley, 1999.
- Philippe Kruchten, *The Rational Unified Process, An Introduction*. Addison-Wesley, 2000.
- Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process* (3rd ed.). Prentice Hall, 2004.
- Robert Martin C. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002.
- Jeffrey Putz, *Maximizing ASP.NET Real World, Object-Oriented Development*. Addison-Wesley, 2005.
- James Rumbaugh, Ivar Jacobson, and Grady Booch, *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

CHAPTER THIRTEEN

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- ▶ Explain the different types of objects and layers in a design
- ▶ Develop communication diagrams for use case realization
- ▶ Develop sequence diagrams for use case realization
- ▶ Develop updated design class diagrams
- ▶ Develop multilayer subsystem packages
- ▶ Explain design patterns and recognize various specific patterns

CHAPTER OUTLINE

- ▶ Object-Oriented Design with Interaction Diagrams
- ▶ Use Case Realization with Communication Diagrams
- ▶ Use Case Realization with Sequence Diagrams
- ▶ Developing a Multilayer Design
- ▶ Updating and Packaging the Design Classes
- ▶ Design Patterns

OPENING CASE NEW CAPITAL BANK: PART 2

The integrated customer account system project for New Capital Bank was now two months old. The first development iteration had gone pretty well, although there were a few snags because the team was still learning the ins and outs of iterative development projects. Bill Santora, the project leader, was discussing some of the system's technical details with Charlie Hensen, one of his team leaders, in preparation for the iteration retrospective.

"How is the team feeling about doing detailed design?" Bill asked Charlie, who was one of the early critics of doing more formal design. "I know some of the programmers wanted to just start coding from the use case descriptions that were developed with the users. They weren't very happy about taking the time to design. Is that still a problem?"

"It really has worked out quite well," Charlie said. "As you know, I was skeptical at first and thought it would waste a lot of time. Instead, it has allowed us to work together better because we know what the other team members are doing. I also think the system is much more solid. We're all using the same approach, and we've discovered that there are quite a few classes we share. Of course, we don't waste a lot of time making fancy drawings. We do document our designs with temporary drawings, but that is about as far as we take it."

"What would you say were the strengths and weaknesses of our approach?" Bill asked, "Or are there ways you think we could do it better in this next iteration?"

"I really like the approach of first doing a rough design using CRC cards," Charlie replied. "It's nice to have a couple of users there with us to verify that our collaborations are correct. For the simple use cases, we can work with the users to lay out the user interface. Between the CRC cards and the user-interface specifications, we should have enough to program from, especially now that we have the basic structure set up. Then, for the more complex use cases, we can go ahead and do a detailed design with communication diagrams or sequence diagrams. The nice thing about these interaction diagrams is that they're detailed enough for us to hand the designs over to some of the junior programmers. It makes them much more effective in their team contributions."

"So, would you change our approach or do you think it's working the right way?" Bill asked, still looking for ways to improve the process.

"Well, it really is working pretty well right now," Charlie said. "One thing I really like about it is that we have a common domain class diagram that everyone can access and review. That really helps when you're ready to insert some code into a class to check and see what is already there. The central repository for all our code and for the diagrams we do formalize is a great tool. I wonder if there is a way to get more use out of that tool. Other than that, I would say let's stick with this approach for another iteration and see if it needs changing after that."

Overview

In Chapter 12, you learned the objective of object-oriented design and the fundamental approach. In Figure 12-3, you learned that design includes information about the classes and about the processes. In Figure 12-8, you learned the approach to (1) select a use case, (2) identify the classes involved and begin the design class diagram, (3) choose a modeling technique to identify the messages and methods, and (4) finalize the class methods in the design class diagram.

This chapter pursues object-oriented detailed design in more depth and formality. This chapter focuses on the foundation principles, which are based on the concepts of use case realization by using UML interaction diagrams and design patterns. You will learn how to use communication diagrams and sequence diagrams to extend the input messages and define internal messages. These techniques will first focus on the problem domain classes and then extend to multilayer design.

The last section of this chapter is a brief introduction to design patterns. As with any engineering discipline, certain procedures have become tried and proven solutions. Even though object-oriented development is a relatively young engineering discipline, it offers standard ways to design use cases that lead to solid, well-constructed solutions. You will learn a few of those standard designs or patterns.

■ Object-Oriented Design with Interaction Diagrams

The discussion of CRC cards in Chapter 12 introduced the idea of collaborating objects to execute various use cases. Design sessions using CRC cards focus on the problem domain classes and their responsibilities. Even though we did add multilayer classes to the CRC cards, the actual details of how all the layers and classes work together was not addressed. Because the CRC card method focuses on simple use cases, the details of the interactions between classes and layers is left to the programmer. This chapter describes in depth the detailed design of all layers of a multilayer system.

In Figure 12-1, there are three objects representing the three layers of a system. Each object has certain responsibilities. The input window object has the primary responsibility of formatting and presenting student information on the screen.

The student object represents the middle layer, or business logic layer, for the use case. This chapter formalizes the process of precisely identifying methods and defining method signatures, particularly for this layer.

The database access object represents the third layer in the multilayer design. It is responsible for connecting to the database, reading the student information, and sending it back to the student object. It is also responsible for writing the student information back to the database when necessary. This object doesn't come from a problem domain class; it is a utility object created by the designer.

Several questions should come to mind as you review a detailed systems design. First, how do all these objects get created in memory? For example, how and when does the student object get created? How about the database access object? Other questions include: Will other objects be necessary? What object represents authentication? What is the life span of each object? Maybe the student object should go away after the update—but what about the database access object?

The method used to extend the process of detailed design is called **use case realization**. In use case realization, each use case is taken individually to determine all the classes that collaborate on it. As part of that process, any other utility or support classes are identified. Care is taken during this process to define the classes so the integrity of the multilayer architectural design is maintained. As the details of the classes are designed—use case by use case—the design class diagram is also updated as necessary.

As you saw in Figure 12-8, in addition to CRC cards, there are two other models that are useful for systems design—communication diagrams and sequence diagrams. Developing interaction diagrams is at the heart of object-oriented design. The realization of a use case—determining what objects collaborate and the messages they send to each other to carry out the use case—is done through the development of an interaction diagram. Two types of interaction diagrams can be used during design: **communication diagrams** and **sequence diagrams**.

Both communication diagrams and sequence diagrams are a type of UML interaction diagram. **Figure 13-1** is a partial class diagram illustrating this fact. It also shows a few composition classes for each diagram class.

Each type of diagram is capable of providing a rigorous method for use case realization. Interaction diagrams, as the name implies, focus on the interactions between the objects that are required to execute a particular use case. The interactions between the objects are called messages. These methods translate directly to the class methods that support the use case. We saw a little bit of this idea with the “responsibilities” that are identified on CRC cards. However, interaction diagrams are more formal and precise in defining these interactions between objects and across layers.

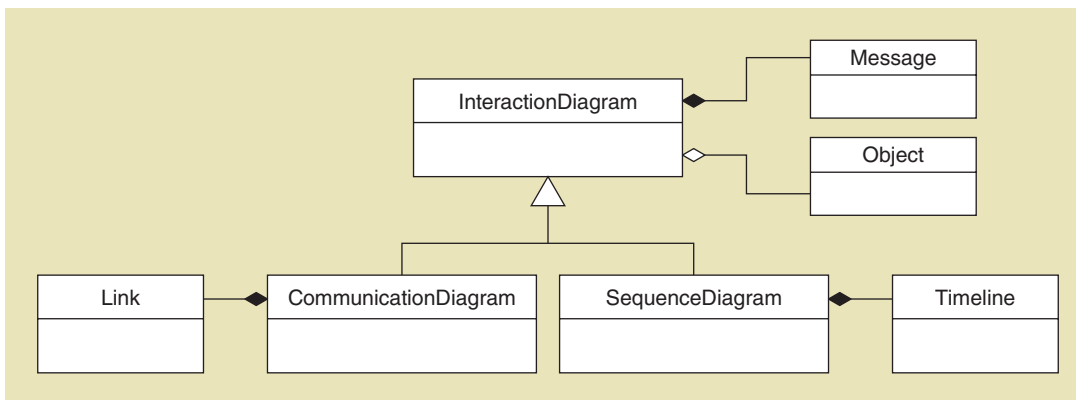
Both communication diagrams and sequence diagrams provide the same basic information. However, communication diagrams are a little less detailed and provide a broad “overview” picture of the interactions. Sequence diagrams

use case realization the process of elaborating the detailed design with interaction diagrams for a particular use case

communication diagram type of interaction diagram that emphasizes the objects that send and receive messages for a specific use case

sequence diagram type of interaction diagram that emphasizes the sequence of messages sent between objects for a specific use case

FIGURE 13-1 Communication diagrams and sequence diagrams are interaction diagrams



are more detailed and allow the designer to visually document the process flow of the messages. This chapter begins the discussion with communication diagrams and then moves to sequence diagrams. First, the following section briefly discusses the concept of a use case controller.

■ The Use Case Controller

The previous chapter briefly introduced the concept of a use case controller in the discussion of CRC cards, but did not elaborate on its use. In reality, a controller class is part of a popular design technique called Model-View-Controller. (This technique is discussed in more detail in the “Use Case Controller” section under “Design Patterns.”)

Let us formalize the concept of a use case controller. For any particular use case, messages come from the external actor to a windows class—that is, an electronic input form—and then to a problem domain class. One issue in systems design is the question of which problem domain class should receive input messages to reduce coupling, maintain highly cohesive domain classes, and protect from variations between the user interface and the domain layer. (Do you remember these good design principles discussed in Chapter 12?) Designers often define intermediary classes that act as buffers between the user interface and the domain classes. These classes are called *use case controllers*. For example, the use case *Fill shopping cart* might have a controller class named *CartHandler*.

A use case controller acts as a switchboard, taking input messages and routing them to the correct domain class. In effect, the use case controller acts as an intermediary between the outside world and the internal system. What if a particular window object needs to send messages to several problem domain objects? Without the use case controller, the input window would need references to all these domain objects. The coupling between the input window object and the internal system would be very high; there would be many connections. The coupling between the user-interface objects and the problem domain objects could be reduced by making a single use case controller object to handle all the input messages. In this way, domain layer design classes can remain more cohesive by focusing only on the precise functions that truly belong to that domain object.

The following examples define a controller class for each use case. This is a common practice, and many development environments (such as Java Struts) automatically define a controller class for each use case. Of course, this creates many artifact objects in a system. If there are 100 use cases, there would be 100 use case controller artifact objects. To reduce the number of controllers, developers sometimes combine the control of several closely related use cases into a single use case controller. Either approach, if done judiciously, provides a

good solution. A use case controller is a completely artificial class created by the person doing the system design. Sometimes, such classes are called *artifacts* or *artifact objects*.

■ Use Case Realization with Communication Diagrams

Adaptive projects that use iteration and Agile modeling techniques minimize the formality of design diagrams. In this chapter (and possibly for your homework assignments), the diagrams will be developed using Microsoft Visio. However, in real projects, hand-drawn diagrams can be scanned and transmitted to all the team members just as easily. Design diagrams are helpful in communicating design decisions throughout the team.

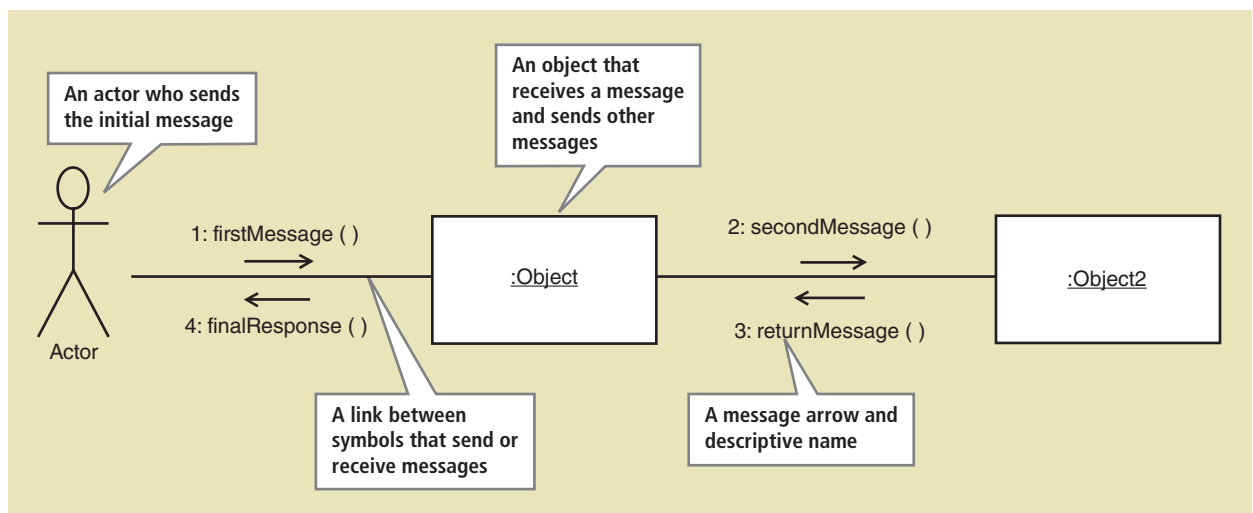
■ Understanding Communication Diagrams

A communication diagram consists of actors, objects, links, and messages. **Figure 13-2** illustrates how these various components are illustrated on the diagram.

These components have the following characteristics:

- **Actor.** This represents an external role of the person or thing that initiates the use case by sending the initial message. The actor will also send any other messages that are external to the system. These messages can come into the system through data that is entered on forms. They can also come electronically by other external devices or systems.
- **Object.** The objects are the instantiated class objects that perform actions to help carry out the use case. In design diagram terms, they receive the messages that request services. Objects can both receive messages and send messages. Note we use object notation—with a colon and underlining—and not class notation. The execution of a use case is performed by instantiated objects and not classes.
- **Link.** Links are simply the connectors that illustrate the flow of the messages. These links do not mean the same as the navigation visibility arrows, although they frequently exist between the same objects. They are only used to show where the messages flow. Messages can flow in either direction on a link.

FIGURE 13-2 Symbols used in a communication diagram



- **Message.** Messages are a primary element of the diagram. A message has an originating object or actor, and a destination object or actor. A message is a request for service. A message can also be a return of data from a previous message. A message will invoke a service in an object. In programming terms, a message is the same as a procedure call or a method call.

Chapter 5 first presented the idea of a message in the discussion of a system sequence diagram. Messages in a communication diagram have the same meaning. However, the syntax of the message is slightly different. The syntax contains five elements, all of which are optional. The format of a communication diagram message is the following:

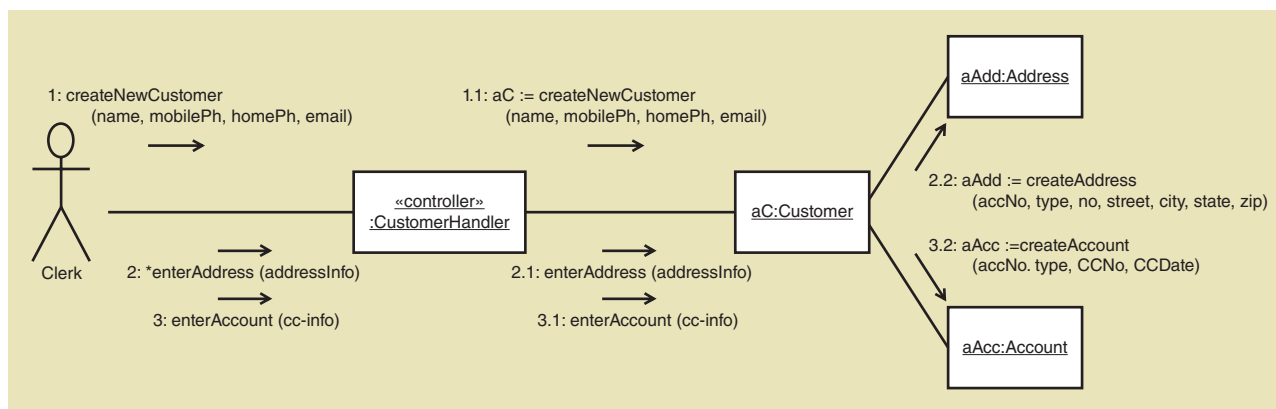
[true/false condition] sequence-number: return-value: = message-name (parameter-list)

- **True/false condition.** This is a condition that is tested for true or false. If it is true, then the message is sent; if false, the message is not sent. It only applies to sending the message. It is optional.
- **Sequence number.** The sequence number is used to identify the order, or sequence, of the messages. They can be simply numbered by integers. Developers also often use hierarchical dot notation (i.e., 1.0, 1.2, 2.1.3, etc.) to denote dependency as well as sequence. For example, if a use case has two separate input messages, each with several subsequent messages that travel the same links, they could be uniquely partitioned by using 1.0 and 2.0 series. Note that a colon follows the sequence number.
- **Return-value.** The return value is a value that the message returns after the completion of the service requested in the forward part of the message. In programming terms, this is similar to a method return value. Return values can be returned either with this format (i.e., as part of the message) or as a separate return message.
- **Message-name.** This is the name of the message. As noted, it usually uses camel case with the initial letter lowercase. To name the message, it is usually best to describe the service that is requested from the destination object. For example, `createAccount` could be a message name that requests the specific service from the destination object. Notice that if a return-value is given, that the two items are separated by a “:=” sign.
- **Parameter-list.** The parameter list contains those items that are being passed to the destination object via the message. Again, in programming terms, these are the arguments that would be passed when a method is invoked.

Let’s look at a typical communication diagram and take note of the information it provides and its characteristics. Remember that a communication diagram is useful to design and document use cases of medium complexity. In other words, this type of diagram works best for use cases that are not too large with many messages. A major benefit of a communication diagram is that it provides a snapshot view of the classes and messages. However, a disadvantage is that there is not much space allowed for messages, so it can easily become cluttered with overlapping information. **Figure 13-3** is a communication diagram for the *Create customer account* use case. Refer back to Figure 12-19 and recall the set of CRC cards created for this same use case. You might want to compare the two solutions, shown in Figure 12-19 and 13-3.

This use case has three input messages: `createNewCustomer`, `enterAddress`, and `enterAccount`. Figure 13-3 shows those three messages as input messages coming from the Clerk actor into the `:CustomerHandler` controller object. Note the sequence numbers indicate the order of the messages. The first message specifies the individual parameters that are to be passed with the message. However, due to space limitations, Figure 13-3 does not show the detail parameters for message 2 or 3. The programmer would have to look at the class diagram and determine what information needs to be passed, or he could provide a footnote to the drawing to provide that additional detail. On the second message, the

FIGURE 13-3 Communication diagram for Create customer account use case



asterisk between the number and the name indicates that the message may be sent multiple times. This is called a multiply occurring message.

The next set of messages, 1.1, 2.1, and 3.1, use the hierarchical dot notation to indicate that they are part of a sequence. Looking at 1.1, notice that it has a return value. Also note that the Customer object is named aC:Customer. The underline indicates an object and not a class. The aC: is the name or identifier for this particular object. That information (i.e., the name of the object) is returned to the :CustomerHandler object. Referring back to Figure 12-16, note that the :CustomerHandler object has navigation visibility to the :Customer object. This mechanism of returning the identifier of the newly created :Customer object provides this navigation visibility. Thus, the :CustomerHandler object can pass the other messages to the correct object because it has a reference to it.

The next set of messages, 2.2 and 3.2, have a similar form. Each sends a create message to the appropriate object and returns the identifier of the newly created object. These two messages include the detail parameters to be passed. In the earlier messages, 2, 2.1, 3, and 3.1 only abbreviated parameter lists were given. The full parameter list is more correct. The abbreviated lists were used purely to save space. Remember, the purpose of modeling is to provide guidelines for programming. So designers sometimes take liberties with the formality of the model, as long as it is still understandable.

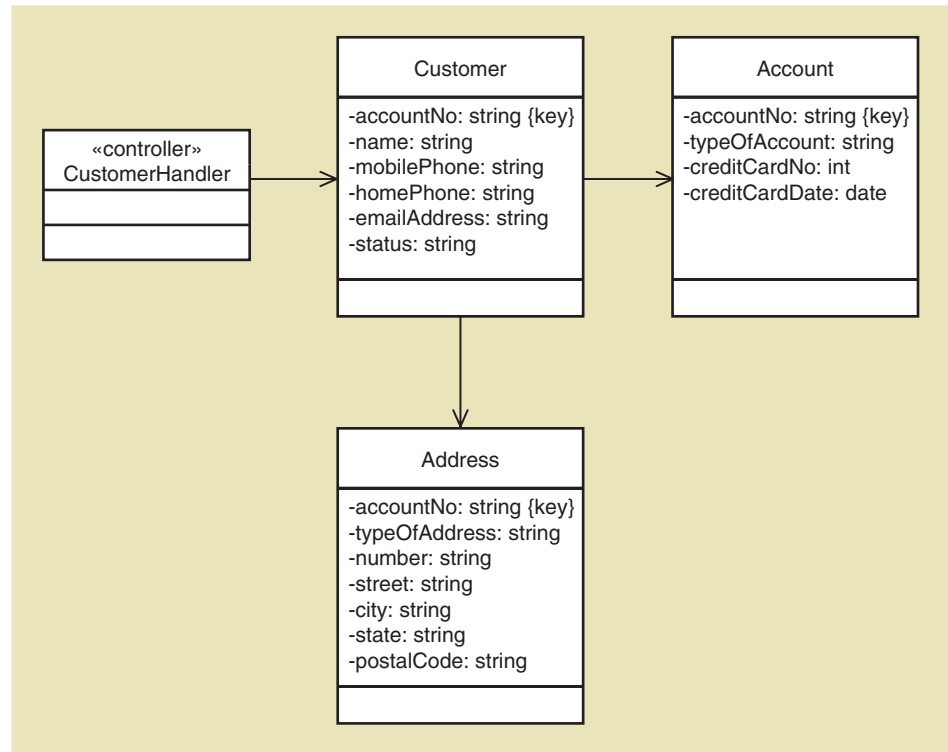
Finally, it is worth mentioning that some return messages have been left off from :CustomerHandler to the Clerk. A system will normally return and display the data that was entered so that the actor can verify that the correct information was stored. In Figure 13-2, the method of returning data is indicated with a return message.

■ Object-Oriented Design with Communication Diagrams

Now that you can read and understand a communication diagram, it's time to learn how to do use case realization using this technique. You will work through the same use case, *Create customer account*. As you work through the example, your understanding of object-oriented design should deepen due to the detailed discussion of the design steps.

Referring back to Figures 12-3 and 12-8, note the models and information that are prerequisite to detailed object-oriented design. First, you should have completed the first-cut design class diagram. This gives you a preliminary idea of what domain classes will be involved and the logical navigation visibility relationships. From the analysis models, it is helpful to have either an activity diagram or a system sequence diagram. In fact, in the ideal world, both would be available along with a detailed use case description.

FIGURE 13-4 First-cut design class diagram for Create customer account



Input Models

A slightly enhanced version of Figure 12-16 is included here as **Figure 13-4**. The selection of which classes to include is purely an estimate at this point because we will not determine all the classes until we proceed with the design steps. However, in this simple example, the preliminary assessment is accurate. Do not worry if you do happen to omit some required classes. It will become evident during the design process if something is missing.

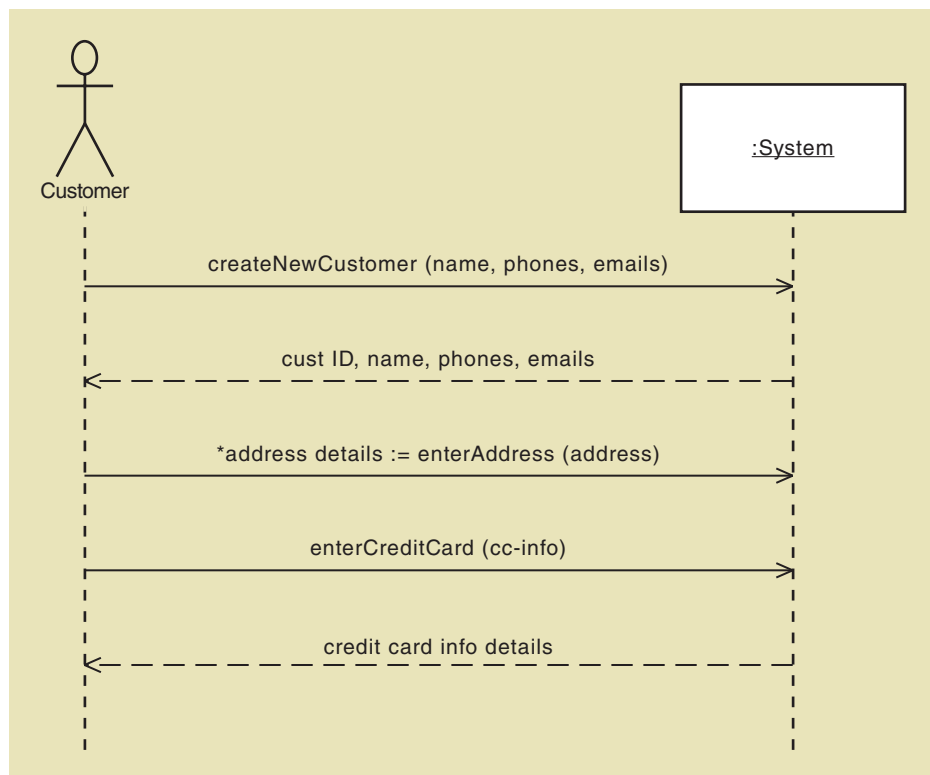
The other source of input information comes from the analysis models. Chapter 5 developed several models for this use case, including a use case description (Figure 5-2), an activity diagram (Figure 5-4), and a system sequence diagram (Figure 5-10). Before initiating the design, you will want to look at all three models to understand as much as possible about the use case. For this example, focus on the system sequence diagram, which is duplicated here as **Figure 13-5**. There are three input messages and two return responses.

Extend Input Messages

For each input message, extend the message to the internal objects within the `:System` object. Follow this process:

1. From the first-cut design class diagram, identify the classes that will be required to carry out the execution of the message. Place the corresponding objects on the diagram.
2. Beginning with the input message, identify each message that will be required for each of the included objects on the diagram. For each message, ensure that the origin object has navigation visibility to the destination object. Determine which object should have primary responsibility for completing the required service. Place appropriate messages based on navigation and responsibility.

FIGURE 13-5 System sequence diagram for Create customer account



© Cengage Learning®

3. Name each message to reflect the service requested from the destination object. Identify and include the parameters that the destination object will require to carry out the requested service. Identify and name any return messages or return values that need to be returned to origin objects.

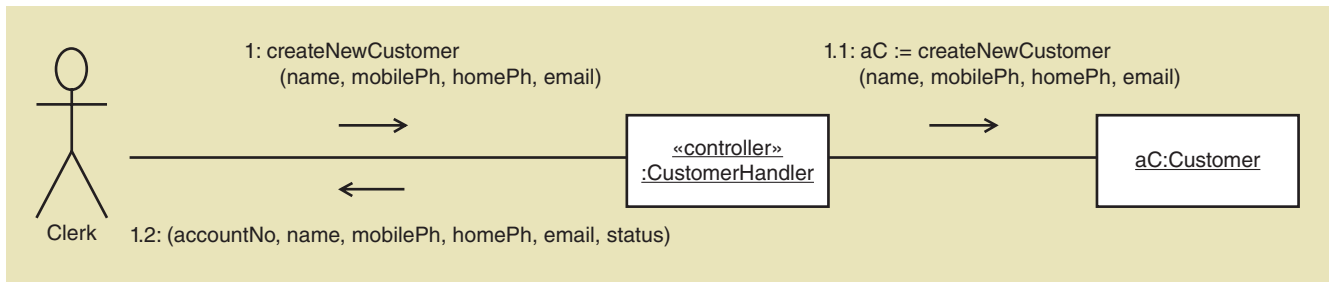
Looking at the first input message, `createNewCustomer (name, phones, emails)`, we decide the following:

1. The required classes for this message are `CustomerHandler` and `Customer`.
2. The required message is `createNewCustomer` and comes from the Clerk to the `:CustomerHandler`. The `:CustomerHandler` then forwards that same message to the `:Customer` object. The `:Customer` object is the appropriate object to carry out this service. The constructor method in the `Customer` class instantiates a new `:Customer` object. This is shown in the diagram with a message directly to the `:Customer` object.
3. The name of the message is appropriate as given in the SSD. The parameters on the SSD, however, do not reflect the attributes of the `Customer` class. The `accountNo` and `status` attributes are both determined by the constructor. The other attributes—`name`, `mobilePhone`, `homePhone`, and `emailAddress`—need to be passed in as arguments. The input parameter list will be modified to reflect these changes.

The return information is a little more complex. First, the identifier, “aC,” of the newly created object is returned from `:Customer` to `:CustomerHandler`. As mentioned previously, this provides navigation visibility from `:CustomerHandler` to `:Customer`. Second, all of the data from the `:Customer` object is returned to the clerk, including the `accountNo` and the `status`. Only the parameter-list is included because all other items of the message syntax are optional, and not required for this message.

Figure 13-6 is the result of these steps. Notice that the messages in the figure are numbered sequentially as they are passed.

FIGURE 13-6 *createNewCustomer* message extended to all objects



Next, you follow the same steps for the second input message, which is **address details := enterAddress (address)*.

Notice several things about this input message. First, it begins with an asterisk. The asterisk means that this message may be sent multiple times. You will carry that forward to the design. Second, the return data is shown right in the message syntax. You have the option of showing it that way, or with a separate message as was done with the first message above. Finally, note that the parameter list is rather abbreviated. We will need to expand it to reflect the attributes of the Address class.

The original name of the message is appropriate for the input message and the second message: *enterAddress*. However, we change the name to *createAddress* for the final name to better reflect service requested. The responsible object for initiating this final request should be the *:Customer* object. Because a customer needs navigation visibility to its dependent objects, including its addresses, it should be the one to invoke the constructor method. The path of the message thus goes from the controller to *:Customer* and then to *:Address*. **Figure 13-7** illustrates the results of extending this message.

The next message, *enterCreditCard (cc-info)*, will follow the same process. We also changed the name of this message to be *enterAccount* because we are creating a new Account object. We also expanded the parameter list to reflect the attributes in the Account class. The other decisions with regard to responsibility and flow of execution are similar to the *enterAddress* message. **Figure 13-8** shows the results of extending the *enterAccount* message.

FIGURE 13-7 *enterAddress* message extended to all objects

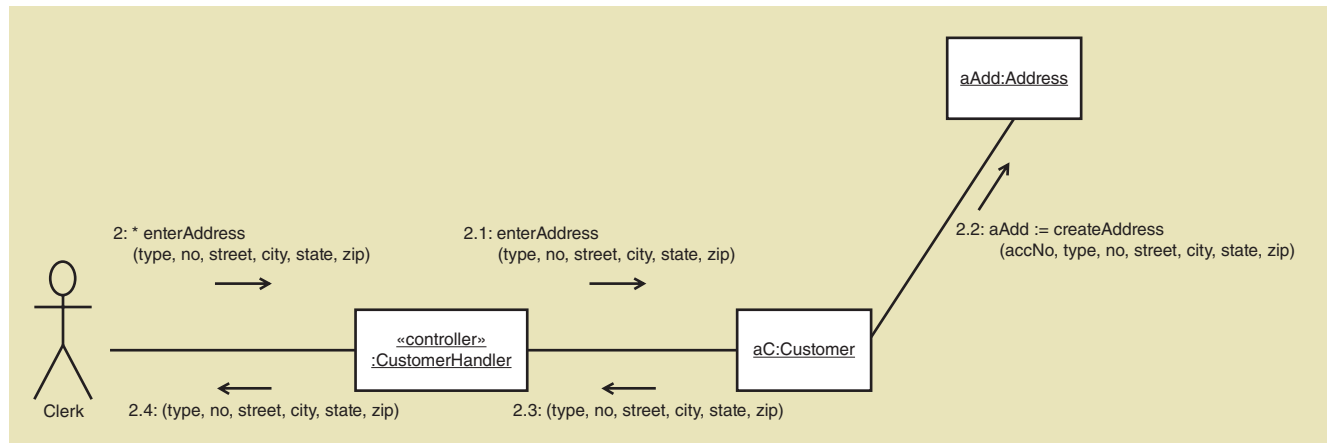
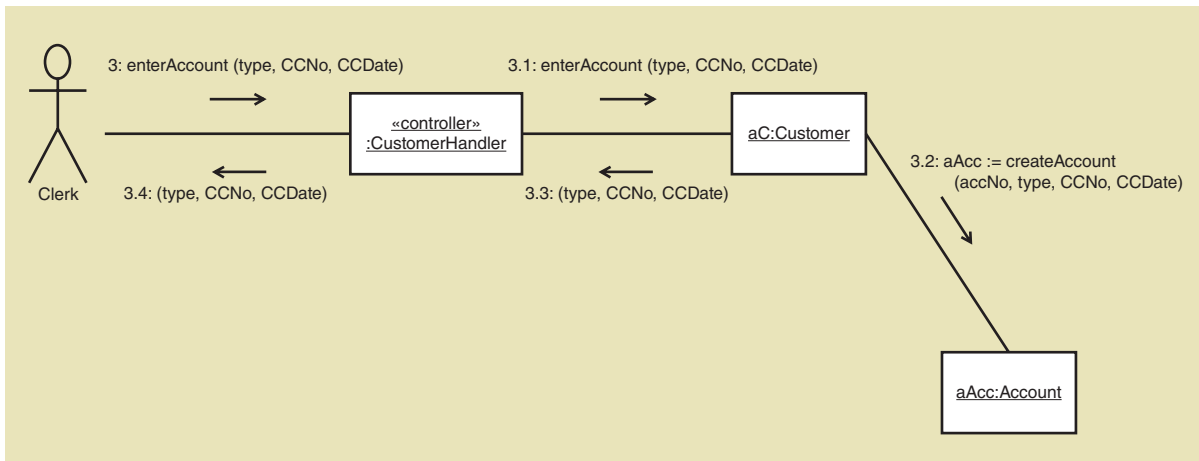


FIGURE 13-8 *enterAccount* message extended to all objects



The final communication diagram for this use case is illustrated in **Figure 13-9**. It is similar to the earlier one in Figure 13-3; however, all of the return messages are included in this final diagram. For ease of reading, the input messages are on top and the return messages are on the bottom. This diagram contains only problem domain objects and does not include view layer or data access layer. The multilayer design using sequence diagrams are covered in the next section. The process is the same for both types of diagrams.

Final Design Class Diagram

As can be seen in Figure 13-9, the messages are specific and precise. Thus, specifying the methods in the design classes can also be precisely defined. In fact, message syntax for communication diagrams is quite similar to the method

FIGURE 13-9 *Final communication diagram for Create customer account use case*

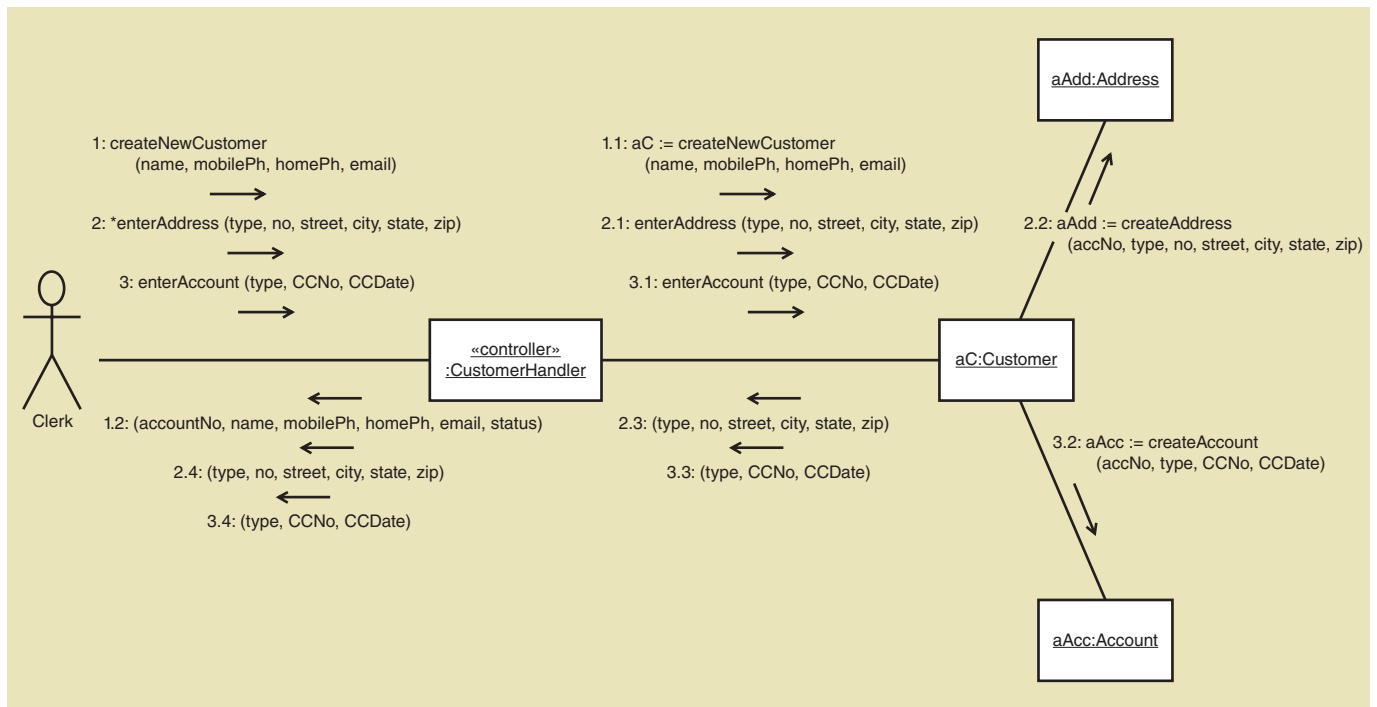
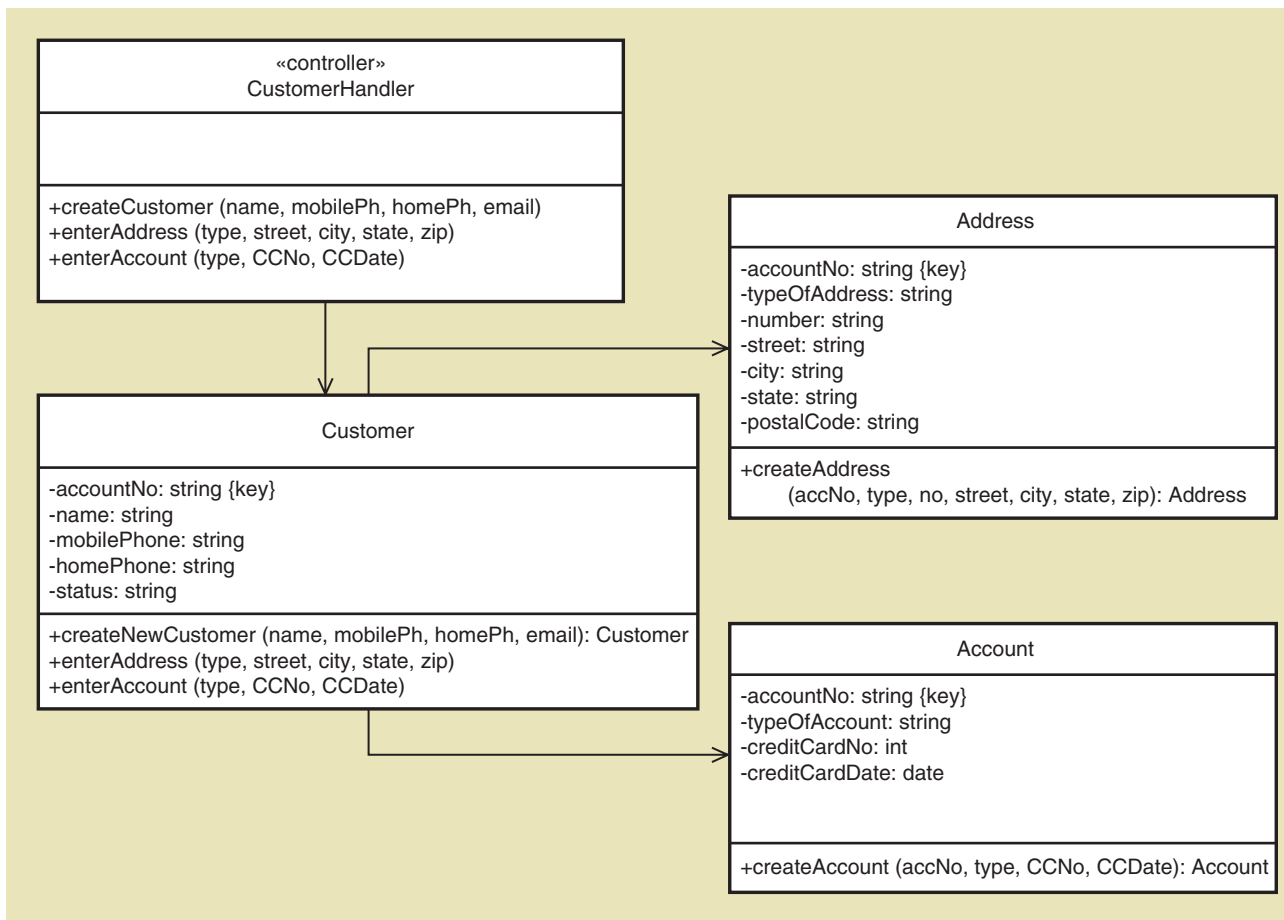


FIGURE 13-10 Design class diagram for methods added from Create customer account



syntax that is used for the design class diagram. **Figure 13-10** illustrates the final design class diagram. You will note that is very similar to Figure 12-22. In this instance, however, the constructor methods for each class are included.

■ Use Case Realization with Sequence Diagrams

Now that you have learned how to do object-oriented design with CRC cards and with communication diagrams, this section explains how to design complex use cases using sequence diagrams. The next section provides a partial sequence diagram to introduce the terms and composition of a sequence diagram. The chapter then demonstrates the process of use case realization by using the use case that we have been working with previously, *Create customer account*. The chapter also demonstrates a more complex example with the use case *Fill shopping cart*. These examples illustrate the core process of organizing and structuring the problem domain classes into the solution for the use case. The final examples explain how to add the data access layer classes and the view layer classes. Each layer is illustrated with a detailed example using the same use case.

■ Understanding Sequence Diagrams

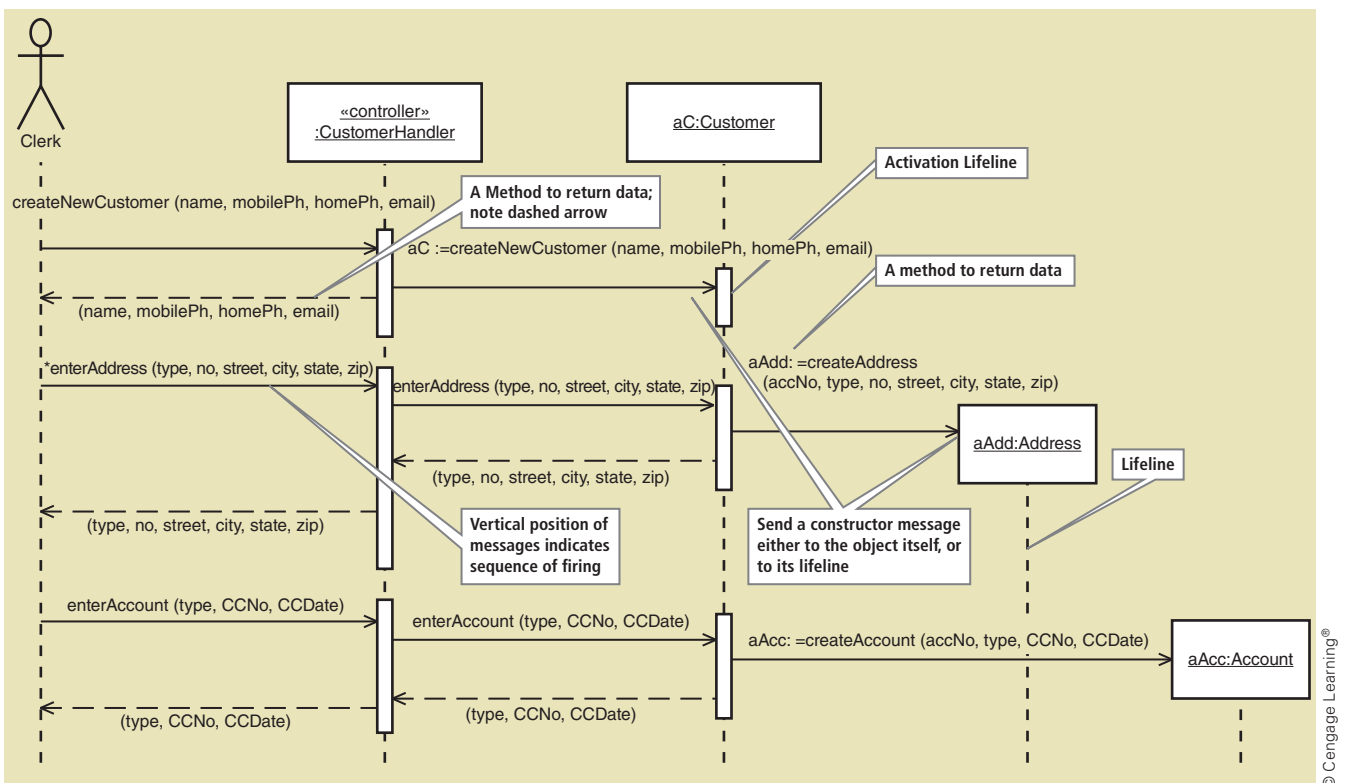
This section first reviews the elements of a sequence diagram to ensure that you remember and understand how to read a sequence diagram. You first learned about sequence diagrams in Chapter 5 when you learned how to develop a system sequence diagram (SSD).

The most important information on an SSD is the sequence of messages between the actor and the system. There may be a single input message or many. The input messages may have data parameters or not. There also may be Loop frames, Alt frames, and Opt frames as well as repeating inputs and outputs. A Loop frame denotes a set of messages within a loop. An Alt frame is similar to an *if-then-else* statement or *switch* statement, which allows the firing of different sets of messages. An Opt frame is an optional invocation of a set of messages. You will see examples of these later in this chapter. First, let's review the basic elements in a sequence diagram.

Figure 13-11 is a sample sequence diagram. In many ways, it is similar to a communication diagram. The actor is an external role, in this case, a Clerk. The boxes are instantiated objects from the corresponding classes. Object notation is used. Some objects are named objects, such as `aC:Customer`. The primary benefit of the sequence diagram is the ability to lay out the messages from top to bottom to emphasize the sequence of firing. Below each actor and object is a lifeline, which is used as an indicator of the life of the object. Messages are attached to the lifeline either as a source point or a destination point. Attached to locations of each lifeline are vertical boxes representing **activation lifelines**. You might consider these activation lifelines the time period when a method is executing.

activation lifeline a representation of the period during which a method of an object is alive and executing

FIGURE 13-11 Sample sequence diagram from Create customer account use case



Some developers use them; others do not. There are two ways to indicate data being returned, either by a return assignment with the “:=” operator, or as a return message using a dashed arrow. Messages that invoke a constructor can be sent either to a lifeline or to the object itself. Both examples are shown in the figure.

As with a communication diagram, when a message is sent from an originating object to a destination object, in programming terms, it means that the originating object is invoking a method on the destination object. Thus, by defining the messages to various internal objects, we are actually identifying the methods of that object. The data that is passed by the messages corresponds to the input parameters of the methods. The return data on a message is the return value from a method. Hence, once a use case is realized with this detailed design process, the set of classes and required methods can be extracted so programming can be completed.

■ Design Process for Use Case Realization

The design process for using sequence diagrams is the same as it is for communication diagrams. The starting point for the detailed design of a use case is always its SSD and the first-cut design class diagram. Other models, such as a use case description and activity diagram, are also helpful. Remember that the SSD only has two lifelines—one for the actor and one for the system. Starting with the SSD, each input message is taken, one at a time, and extended to all of the internal classes so that the desired result is obtained. Any data to be returned is identified and added. Figure 13-11 is the sequence diagram solution of the *Create customer account* use case. It was developed using the same three steps that were used to extend communication diagram messages. You will notice that it has the same set of messages, but displayed differently.

Let us analyze this solution based on some of the principles of good design that were discussed in Chapter 12. The use case controller provides the link between the internal objects and the external environment. The responsibilities assigned to :CustomerHandler are to catch incoming messages, distribute them to the correct internal domain objects, and return the required information to the external environment.

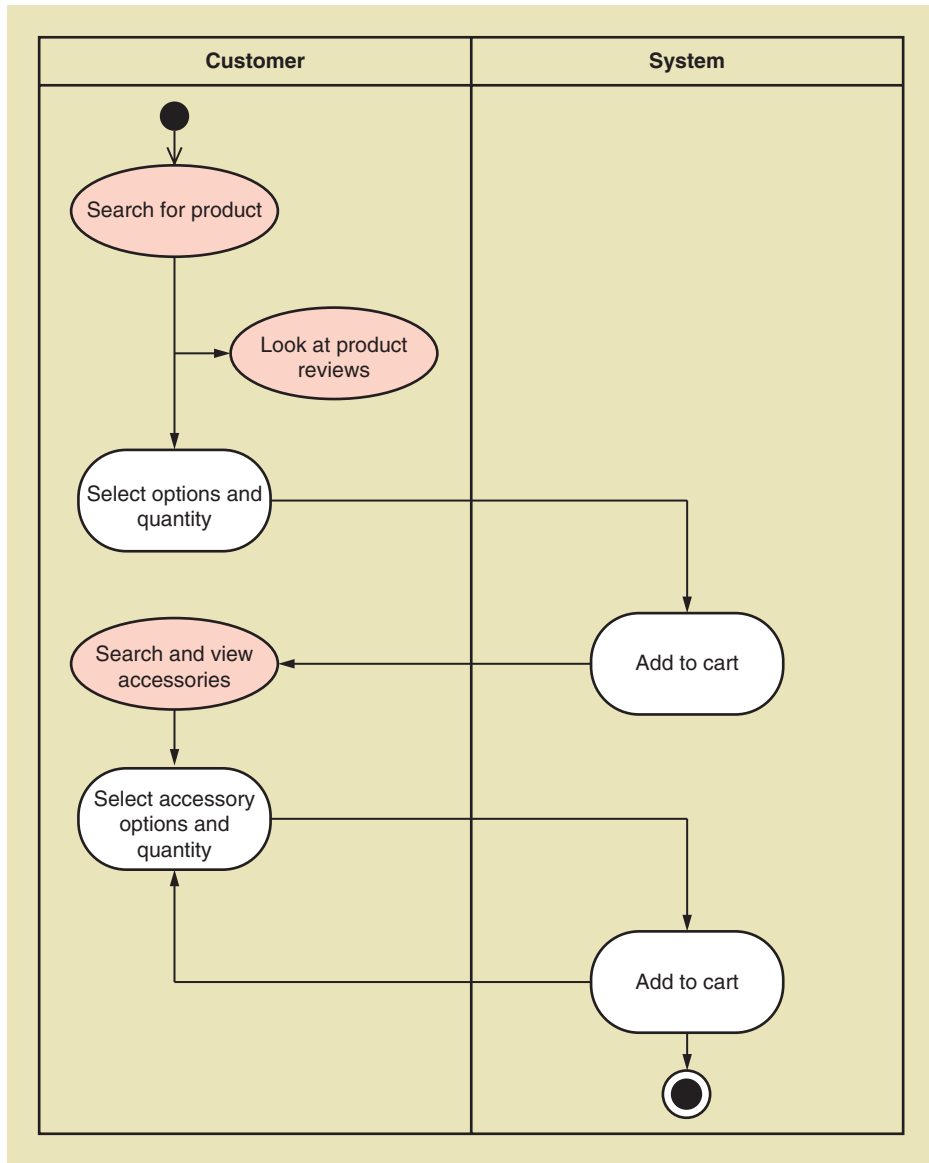
The responsibility assigned to :Customer is to be in charge of creating itself and to control the other required updates to subordinate objects. The :Address and :Account objects create themselves. Coupling is straightforward, being basically vertical on the hierarchy. Thus, the assignment of responsibilities and corresponding messages conforms to good design principles. Other issues will need to be addressed as the design expands to include three layers.

■ Sequence Diagram: *Fill Shopping Cart* Use Case

This section looks at a slightly more complex example of a sequence diagram. With this example, you will see the strength of sequence diagrams in modeling complex use cases. **Figure 13-12** is an activity diagram for the *Fill shopping cart* use case. You will remember from Figure 3-15 that this use case “included” three other use cases, as shown in Figure 13-12. By designing the use case in this manner, with other use cases included, our solution will only have to focus on those functions that actually add items to the shopping cart.

The SSD for this use case is quite simple. **Figure 13-13** shows that there are only two input messages to the system: adding an item and adding an accessory item. As you analyze the SSD, notice that adding an item to the shopping cart and adding an accessory to the cart are the same operation. The only difference is that adding accessories requires a loop for any multiple accessories added for that same item. Because this is the only difference, we can simplify the

FIGURE 13-12 Activity diagram for the Fill shopping cart use case

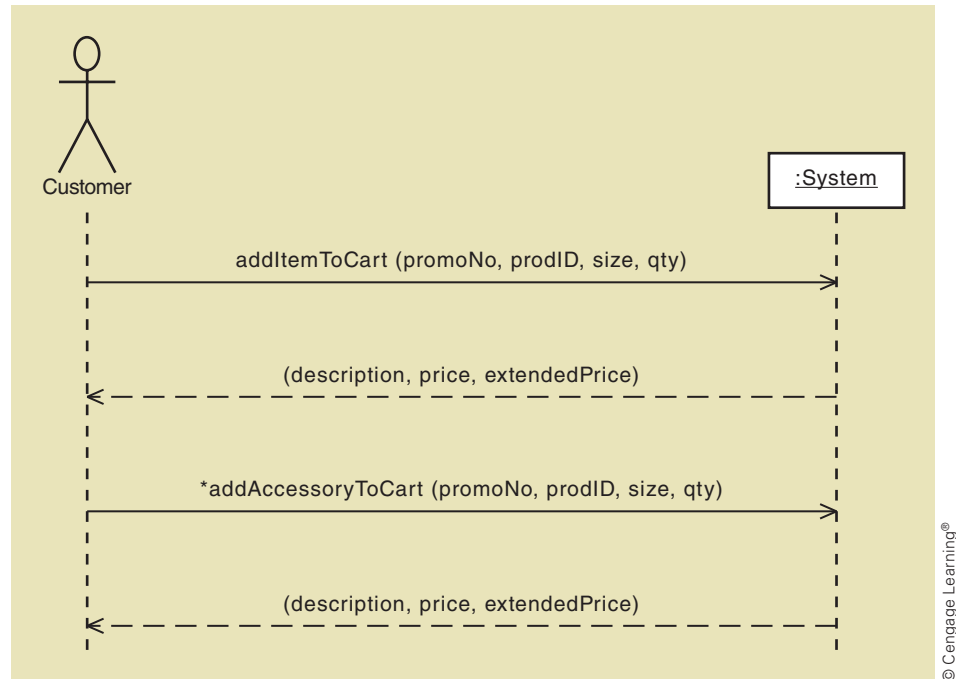


© Cengage Learning®

diagram by limiting the solution to the first message. (Note: An additional class, *AccessoryPackage*, is required for the use case *Search and view accessories*, but because we aren't designing that use case, it isn't required for this solution.)

We begin this design by developing the first-cut design class diagram. Refer back to Figure 4-23, which presented the class diagram for the CSMS sales subsystem. Using that diagram, we can identify the classes that are required for this use case. The *Customer*, *Cart*, and *CartItem* classes are necessary because the use case will be adding items for this customer to the customer's cart. To create a cart item, the system will need to know what product it is, if there are items in stock, and the price for the item. Therefore, other classes that are required are *InventoryItem*, *ProductItem*, and *PromoOffering*. As we develop the solution, we may have to add classes, but this appears to be sufficient for now. Navigation visibility between these classes will be from the controller to the *Customer* class and to the *Cart* class once it has been created. The *Cart* class will be able to access the *CartItem* class. The *CartItem* class should have visibility to the other classes, such as *ProductItem*

FIGURE 13-13 System sequence diagram for Fill shopping cart use case



© Cengage Learning®

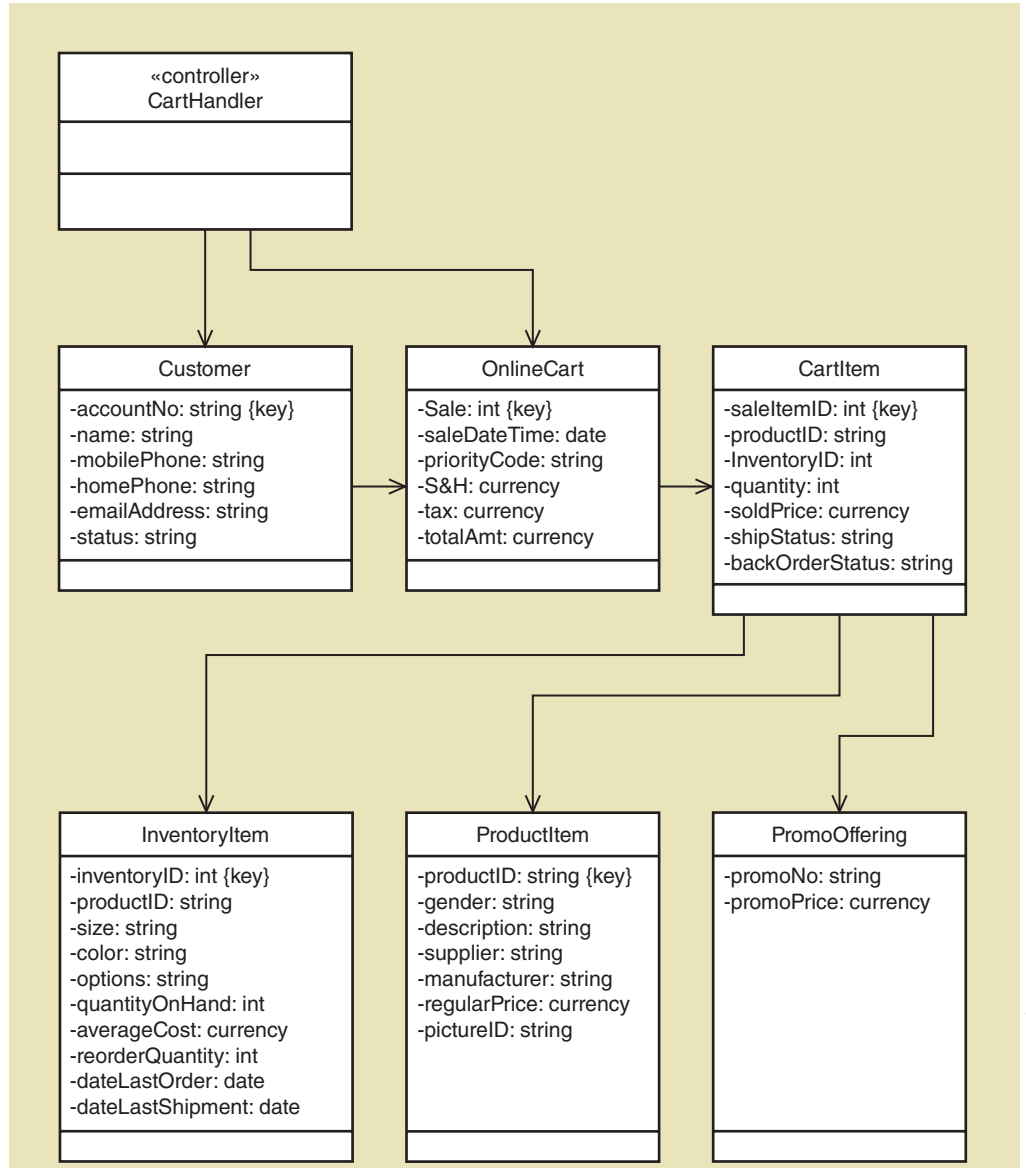
and InventoryItem that contain the necessary information. **Figure 13-14** shows the first-cut design class diagram, which includes several additions to the domain model. In Figure 13-14, several key identifier fields have been added to some classes. One correction to the domain model was also made. In Chapter 9 during the normalization process the regularPrice attribute was moved out of the PromoOffering class and into the ProductItem class, which is where it correctly belongs.

To design this use case, we will follow the same steps that were outlined in the “Extend Input Messages” section for use case realization using communication diagrams. Even though this use case has fewer input messages than the previous example (*Create customer account*), the processing required to execute a message is more complex.

Let’s take the first input message, addItemToCart. This message first comes to the CartHandler. Obviously, it is not possible to add an item to a cart if there is no cart. So how do you create a cart? One solution is to require the user/actor to create a new cart. This is a poor idea. It is never a good idea to require the user to perform an act if the system can do it automatically. The system can be made smart enough to know if it needs to create an online cart first. How can the system know? There are several ways to design this process. Let’s use the perfect technology assumption that the user has logged on. So :CartHandler should have a reference to the :Customer object. However, if there is no reference to an online cart object, then it needs to be created. The next question is, what object should create the new online cart object? From the domain model, you can see that an online cart object has a cardinality of exactly one to a customer object. So a cart cannot exist without a customer, and from the design class diagram, you can see that the customer must have visibility to the online cart. Hence :Customer is the obvious object to create the :OnlineCart object.

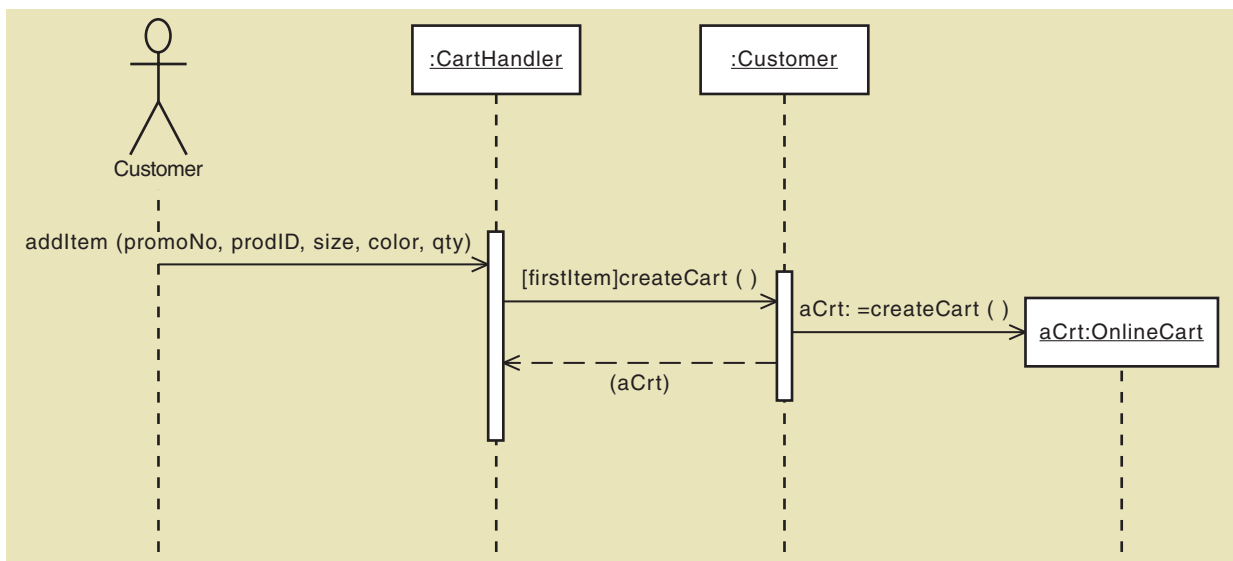
Figure 13-15 shows this first step. The :CartHandler checks to see if it is the first item, and if so, sends a createCart message to :Customer. Notice the true/false condition on that message. The :Customer object sends a create message directly to :OnlineCart to instantiate a new cart. A reference to the new cart is returned first to :Customer, which then forwards it on to :CartHandler. Now both have navigation visibility to the newly created online cart. Figure 13-15 shows both methods of returning data.

FIGURE 13-14 First-cut design class diagram for the Fill shopping cart use case



© Cengage Learning®

FIGURE 13-15 First step in extending <code>addItem</code> message

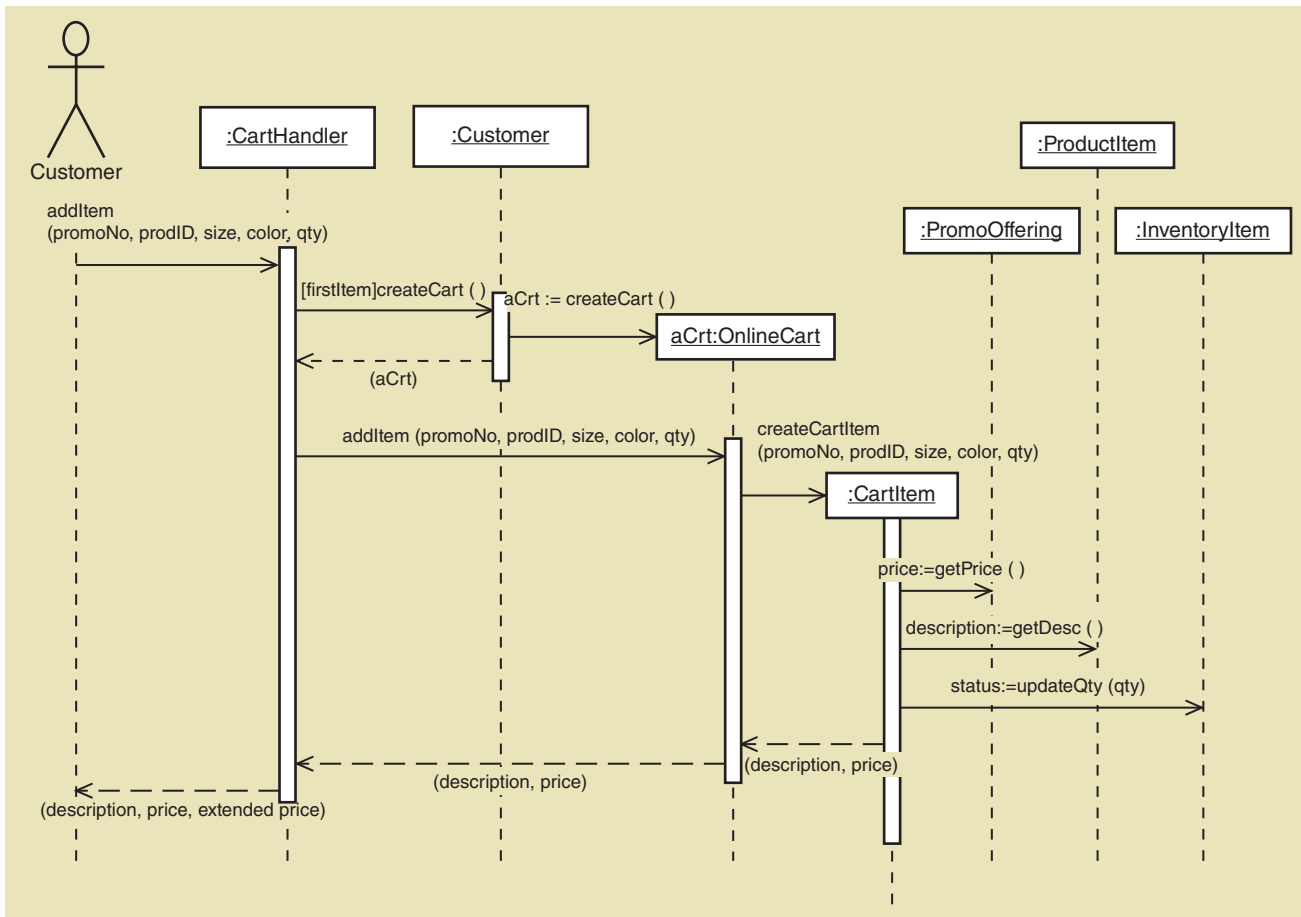


Now that the cart is created, the system can add the item to the cart. Also, notice that if it is not the first item, then this sequence of messages is skipped and the system goes directly to adding the item to the cart. Because `:CartHandler` now has visibility to `:OnlineCart`, it sends a message directly to the cart to add the appropriate item. `:OnlineCart` creates a new cart item and passes the data to `:CartItem` to instantiate a new cart item. `:CartItem` retrieves the necessary data to complete its attributes. We are assuming in this case that the price is the promotional price. Once the cart item has created itself, it returns information back to `:OnlineCart`, which returns it to `:CartHandler` and eventually to the user/customer. **Figure 13-16** illustrates this next set of messages.

As you can see, this simple input message extends to quite a bit of internal activity. How did we know what to do? We observed the attributes of `:CartItem` and noted what classes contained the information that was needed. `:CartItem` is responsible for obtaining the data to instantiate itself. It simply went to the necessary objects to obtain that data.

Referring back to Figure 13-13, note that there is another input message, `addAccessoryToCart (...)`, with the same input parameters. As you think about this, note that adding an accessory is the same activity as adding an item. Accessories are just items that are associated with products for promotions to the customer. Once an accessory item is selected, it simply becomes an item. Hence, the same set of internal messages are required. The only difference is that adding accessories is an optional activity. In Chapter 5, Figures 5-8 and 5-9 showed you how to indicate a loop of messages or an optional set of messages. In this case, the

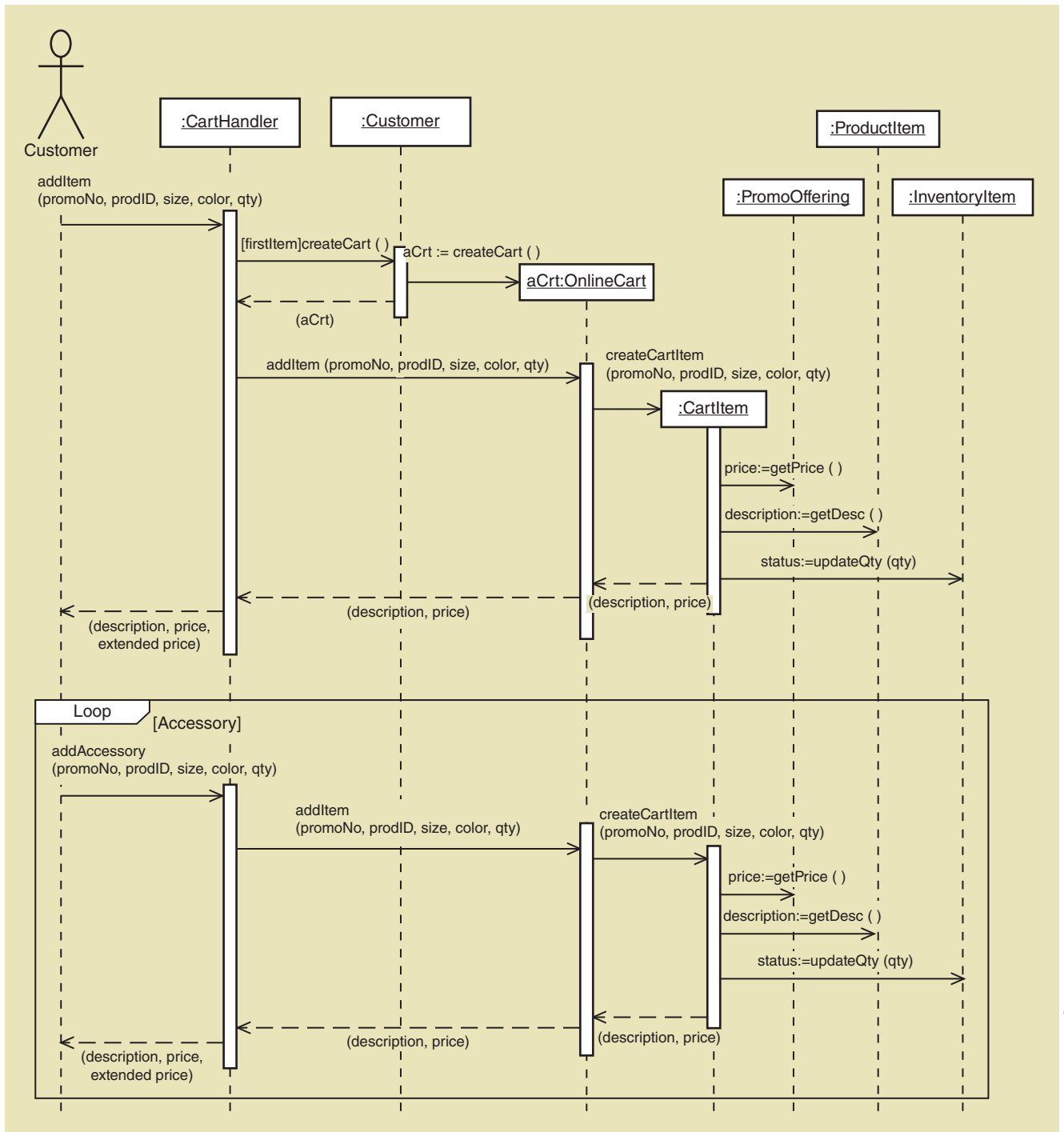
FIGURE 13-16 Completion of Additem message



SSD indicates a repeating message for the accessories. **Figure 13-17** is the full use case with both sets of messages shown, one set for an item and a repeating set for the accessories. The repeating messages inside the Opt, Loop frame are the same as those developed for the first input message, but without the createCart message.

As you identify the specific messages, along with source and destination and the passed parameters, you need to consider some critical issues. As before, an important question is: Which object is the source or initiator of a message? If the message is a query message, the source is the object that needs information.

FIGURE 13-17 Sequence diagram for Fill shopping cart use case



If the message is an update or create message, the source is the object that controls the other object or that has the information necessary for its creation.

Another important consideration is navigation visibility. To send a message to the correct destination object, the source object must have visibility to the destination object. Remember that the purpose of doing design is to prepare for programming. As a designer, you must think about how the program will work and consider programming issues. Given these two considerations and the source considerations discussed in the previous paragraph, we have determined that the following internal messages will be required. For each message, a source object and a destination object have been identified.

- **createCart()**. The `:CartHandler` object will know whether it has received earlier messages to add items. If it hasn't, it knows it must tell the `:Customer` object to create a cart.
- **aCrt:=createCart()**. The `:Customer` object owns the `:OnlineCart` object.
- **addItem()**. A forwarded version of the input message from `:CartHandler` to `:OnlineCart`. Because `:CartItem` objects are dependent on a cart, `:OnlineCart` is the logical object to create `:CartItem` objects. The controller has visibility to `:OnlineCart` from the previous return message, when `aCrt` was returned.
- **createCartItem()**. The internal message from `:Cart` to `:CartItem`. Because `:CartItem` will be responsible for obtaining the data for its attributes, it needs visibility to `:PromoOffering`, `:ProductItem`, and `:InventoryItem`. As a result, information to find those objects is sent as parameters.
- **getPrice()**. The message to get the price from the `:PromoOffering` object. The `:CartItem` initiates the message.
- **getDescription()**. The message initiated by `:CartItem` to get the description from `:ProductItem`.
- **updateQty(qty)**. The message that checks for sufficient quantity on hand. This message also initiates updates of the quantity on hand. `:CartItem` initiates the message.

■ Guidelines and Assumptions Sequence Diagram Development

From the two previous examples, we can distill several guidelines that can help you develop a design for a use case or scenario using sequence diagrams. Several assumptions are also implicit in this process.

■ Guidelines

Designing a use case or scenario by using sequence diagrams involves performing these tasks:

- Take each input message and determine all the internal messages that result from that input. For each message, determine its objective. Determine what information is needed, what class needs it (the destination), and what class provides it (the source).
- As you work with each input message, identify the complete set of classes that will be affected by the message. In other words, select all the objects from the domain class diagram that need to be involved. In Chapter 5, you learned about use case preconditions and postconditions. Any classes that are listed in either the preconditions or postconditions should be included in the design. Other classes to include are those that are created, classes updated during the use case, and those that provide information used in the use case.
- Flesh out the components for each message; that is, add iteration, true/false conditions, return values, and passed parameters. The passed parameters should be based on the attributes found in the domain class diagram. Return values and passed parameters can be attributes, but they may also be objects from classes.

These three steps will produce the preliminary design. Refinements and modifications may be necessary; again, we are focusing only on the problem domain classes involved in the use case.

■ Assumptions

The development of the first-cut sequence diagram is based on several assumptions, including:

- **Perfect technology assumption.** You first encountered this assumption in Chapter 5, when identifying business events. The assumption continues here. You don't include messages such as the user having to log on.
- **Perfect memory assumption.** You might have noticed our assumption that the necessary objects were in memory and available for the use case. We didn't ask whether those objects were created in memory. We will change this assumption when we get to multilayer design. In multiple-layer design, we do include the steps necessary to create objects in memory.
- **Perfect solution assumption.** The first-cut sequence diagram assumes that there are no exception conditions. No logic is included to handle a situation in which the requested catalog or product isn't found. More serious exception conditions, such as the failure of a credit check, might also be encountered. Many developers design the basic processing steps first and then add the other messages and processes to handle the exception conditions later. We do the same here.

■ Developing a Multilayer Design

So far in the development of the sequence diagram, we have focused only on the classes in the problem domain layer. In many instances, this may be sufficient documentation to program the solution—either by yourself or with another programmer. Once you have a solid design for the problem domain classes, adding the view layer and the data access layer is a straightforward, if time-consuming, process. Conforming to the principles of Agile modeling, we don't want to create diagrams unless there is real benefit. We also don't normally keep the design diagrams as documentation because over time, the system will be modified and the diagrams will become obsolete. As Agile modeling suggests, be prudent in the development of models. However, there are times when it is important to see the total picture and identify the need and use of the view layer classes and the data access layer classes. A system developer needs to know how to do complete design for those instances when it is necessary.

Every system will need view layer classes to represent the input and output screens for the application. Data access layer classes aren't always required. The data access layer is required when the business logic is fairly complex and should be isolated from the SQL statements that access the database. The CRC cards example in Chapter 12 for the *Create customer account* use case showed an example of a three-layer design with both view and data access classes (see Figure 12-21). Because CRC cards is an informal technique, we simply added the classes without much discussion of how they are to be connected to the problem domain classes. The following sections develop that same example but with a sequence diagram. This more rigorous modeling technique highlights the issues involved with three-layer design. First, we add the view layer, then the data access layer.

■ Designing the View Layer

In Chapter 8, you learned how to do the design of the user interface. In reality, user-interface design is both an analysis and a design activity. Quite a bit of discovery and understanding is being developed as the systems analysts and the users work together. However, design work is also being done because the actual detailed layout, including input and output data fields, is being developed. You might ask, “If the user interface is already designed, do I need to add the view layer to the sequence diagram?” The answer is a definite maybe.

One advantage of adding it to the sequence diagram is that the programmer can see how the view layer classes integrate with the rest of the design. It becomes a check to make sure the design is correct and complete. It is a good practice to verify several use cases to ensure that the developers understand how the view layer represents the user-interface design and how all the elements integrate together for a smooth program execution. Thus, input for view layer design includes the use case description, the SSD, the activity diagrams, the first-cut design class diagram, and, finally, the user-interface layouts or mock-ups.

User-interface design and the integration of the view layer into a sequence diagram are made even more complex by the fact that many systems require both a Web-based interface and an internal, network-based interface. Fortunately, browsers are becoming more sophisticated, so many new systems can now be designed for only one type of interface. Designing a system with multiple user interfaces is a complex endeavor.

Let's return to the *Add customer account* use case and add the view layer. Remember from the discussion of communication diagrams that there are three input messages: `createNewCustomer`, `enterAddress`, and `enterAccount`. Let's assume that, from the results of our user-interface design, a separate input form for each message is required. Let's add those classes to the sequence diagram. The starting point is Figure 13-11, which is the sequence diagram with the problem domain classes. The process to add the view layer is simply to add an object for each input form or screen. First add the appropriate messages from the Clerk to the input «view» objects. In this instance, the messages are the ones identified from the SSD, and which were used in Figure 13-11. Additional messages are added for the `:CustWindow` to open the `:AddrWindow`, and the `:AddrWindow` to open the `:AcctWindow`. **Figure 13-18** illustrates the resulting diagram with both the domain and view layers.

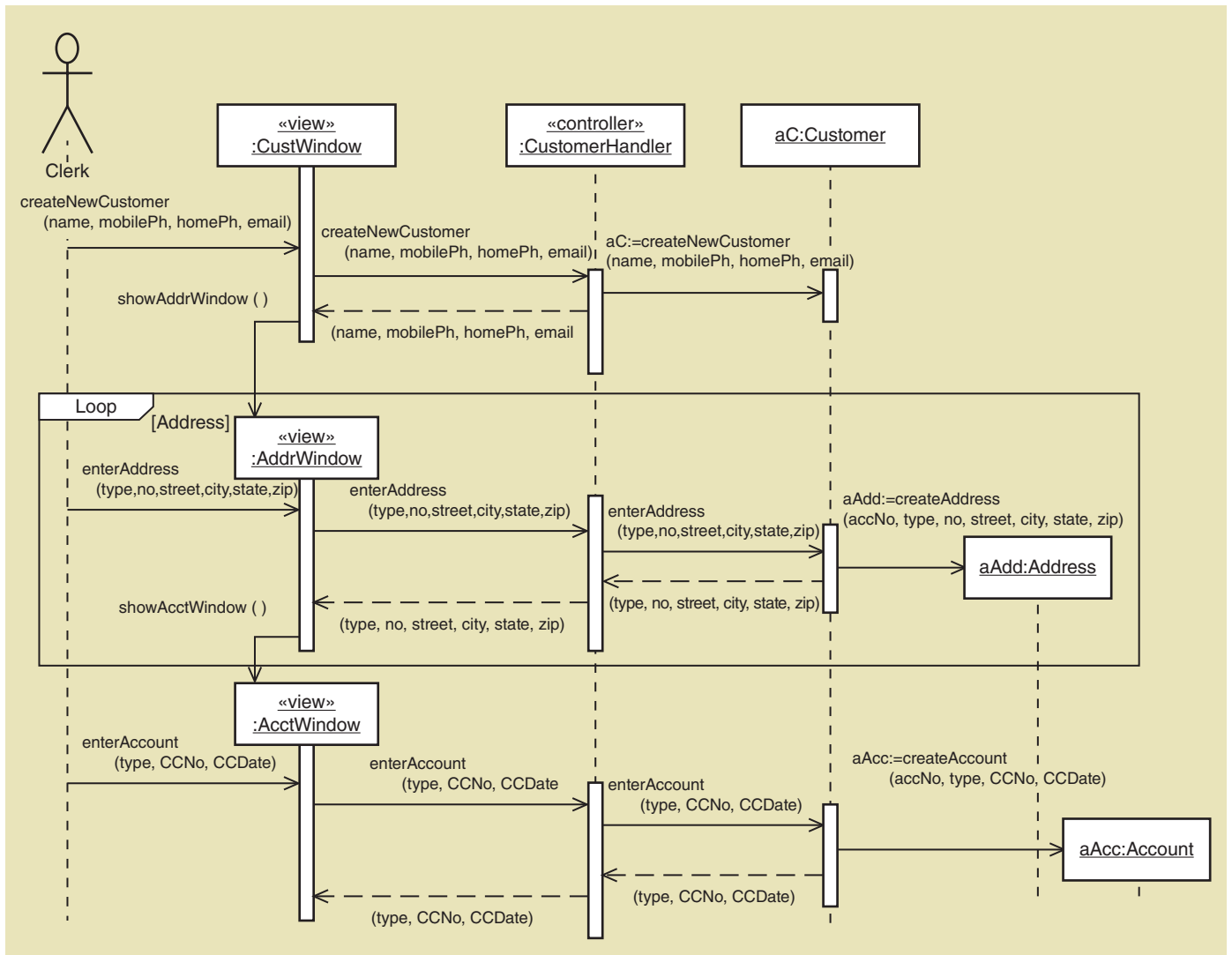
In Figure 13-18, we just added a single view object for each message. This approach was rather straightforward. If we look at the other use case we have been using, *Fill shopping cart*, we have a more complex example. Remember that the *Fill shopping cart* use case included other use cases for *Search for item* and *View accessory combinations*. Obviously, all those use cases go together for a rich and efficient user experience. In **Figure 13-19**, we have added the two view layer objects for searching items and viewing accessories. The first input message, `addItem ()`, will go through the `:SearchItemWindow` object. In other words, when the customer finds something he or she likes, he or she will initiate adding it to his or her cart from that window. The message then causes a detailed `:AddItemWindow` object to display and show the details to verify the addition to the cart. This later window will forward the message on to `:CartHandler`.

Once the item has been added to the cart, another window displays that shows the results of adding this new item. Depending on the design of the user interface, this window might show the single newly added item or it might also show the total shopping cart.

The next three view layer objects, `:ViewAccessWindow`, `:AddAccessWindow`, and `:DisplayItem+AccessWindow`, function in a manner similar to the other view layer objects. The only difference is that the data includes the item and the accessories that have been added to the online cart.

Adding the view layer to your design is a good way to verify that the user interface that was developed with the users is consistent with the application design. All the input messages that were identified and documented on SSDs must be handled by the user interface. If there are messages without input windows or windows without messages, you will know that part of the design is incomplete and that more definition is required.

FIGURE 13-18 Add customer account use case with view layer added

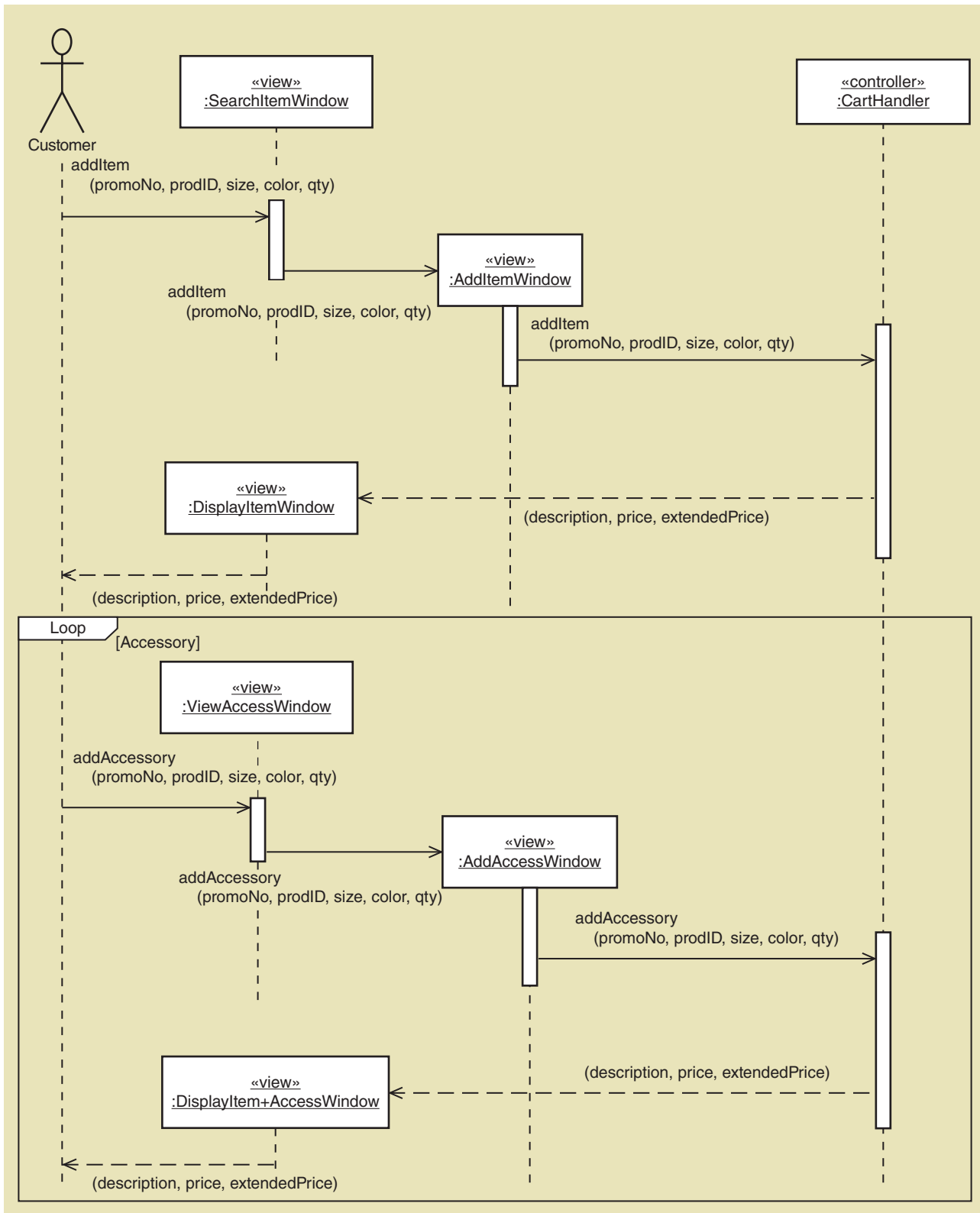


■ Designing the Data Access Layer

The principle of separation of responsibilities is the motivating factor behind the design of the data access layer. On large, complex systems, designers create three-layer designs, including classes whose sole responsibility is executing database SQL statements, getting the results of the query, and providing the information to the domain layer. As hardware and networks became more sophisticated, multilayer design was used to support multitier networks in which the database server was on one machine, the business logic was on another server, and the user interface was on several desktop client machines. This way of designing systems creates more robust and more flexible systems.

In most cases, problem domain classes are also persistent classes, which means that their data values must be stored by the system even when the application isn't executing. The whole purpose of a relational database is to provide this ability to make problem domain objects persistent. Executing SQL statements on a database enables a program to access a record or a set of records

FIGURE 13-19 Partial sequence diagram for the Fill shopping cart use case with view layer



© Cengage Learning®

from the database. One of the problems with object-oriented programs that use relational databases is that there is a slight mismatch between programming languages and database SQL statements. For example, in a database, tables are linked through the use of foreign keys (see Figure 9-9), such as a cart having a

CustomerID as a column so the order can be joined with the customer in a relational join. However, in object-oriented programming languages, the navigation is often in the opposite direction (i.e., the Customer class may have an array of references that point to the OnlineCart objects, which are in computer memory and are being processed by the system). In other words, design classes don't have foreign keys.

This chapter takes a somewhat simplified design approach in order to teach the basic ideas without getting embroiled in the complexities of database access. Let us assume that every domain object has a table in a relational database. (More complex situations exist in which tables must be combined to provide the correct set of objects in memory.)

When a new persistent object is created in memory, the constructor method often initiates the process to write it to the database. When an object is updated, it also needs to be written to the database. The common method to do that is simply to send a message to the data access object. Either the set of object attributes can be sent as parameters or simply as a reference to the object itself. The data access method can pull out the attributes, format an SQL insert or update statement, and write it to the database. **Figure 13-20** is an enhancement of Figure 13-18 with the data access layer added. It is a straightforward addition to add three data access objects and have the newly created objects send messages to write themselves out to the database. Although this diagram is fairly busy, it is organized with the view layer on the left, the problem domain layer in the middle, and the data access layer on the right.

FIGURE 13-20 Create customer account use case with view layer and data layer

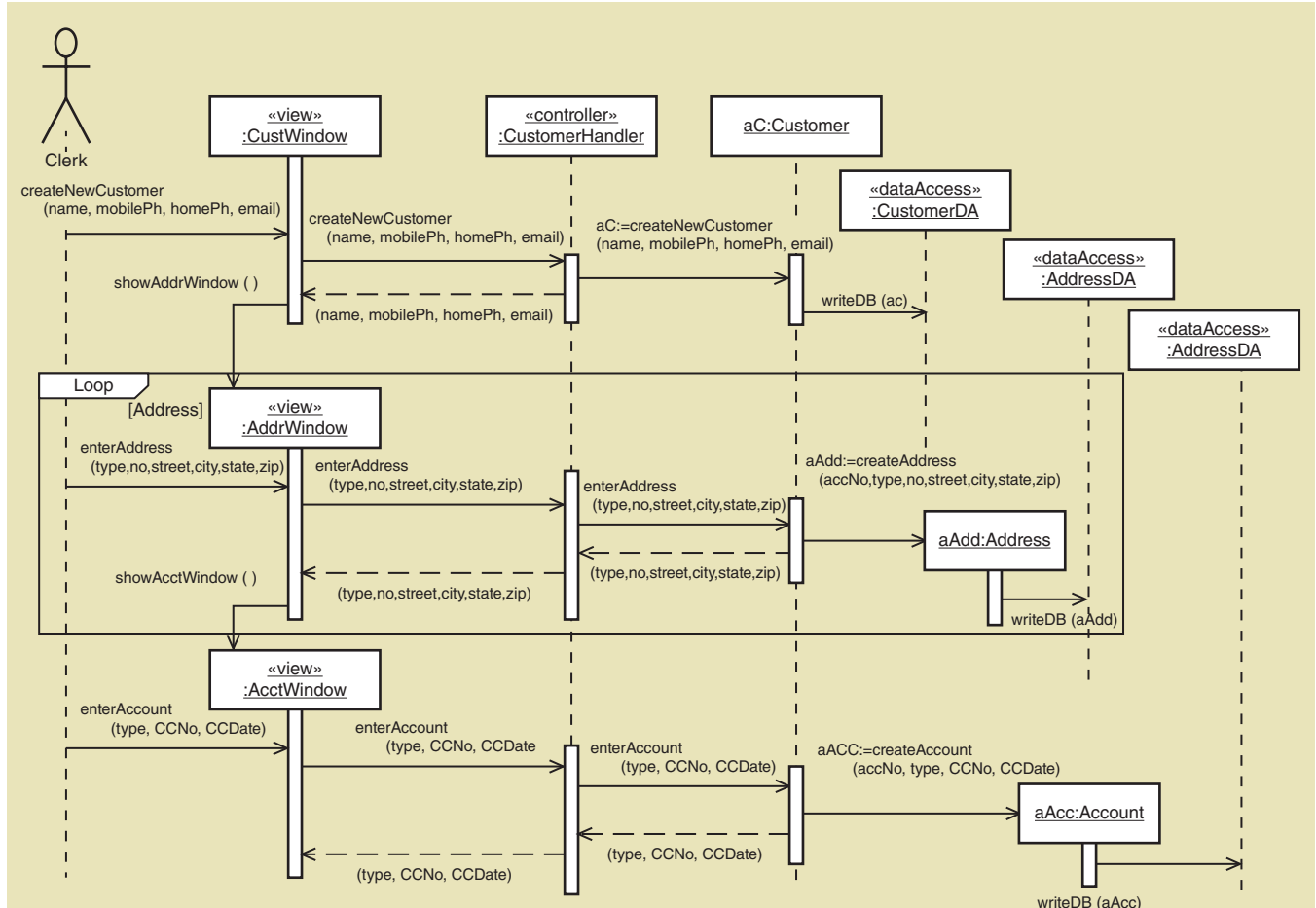
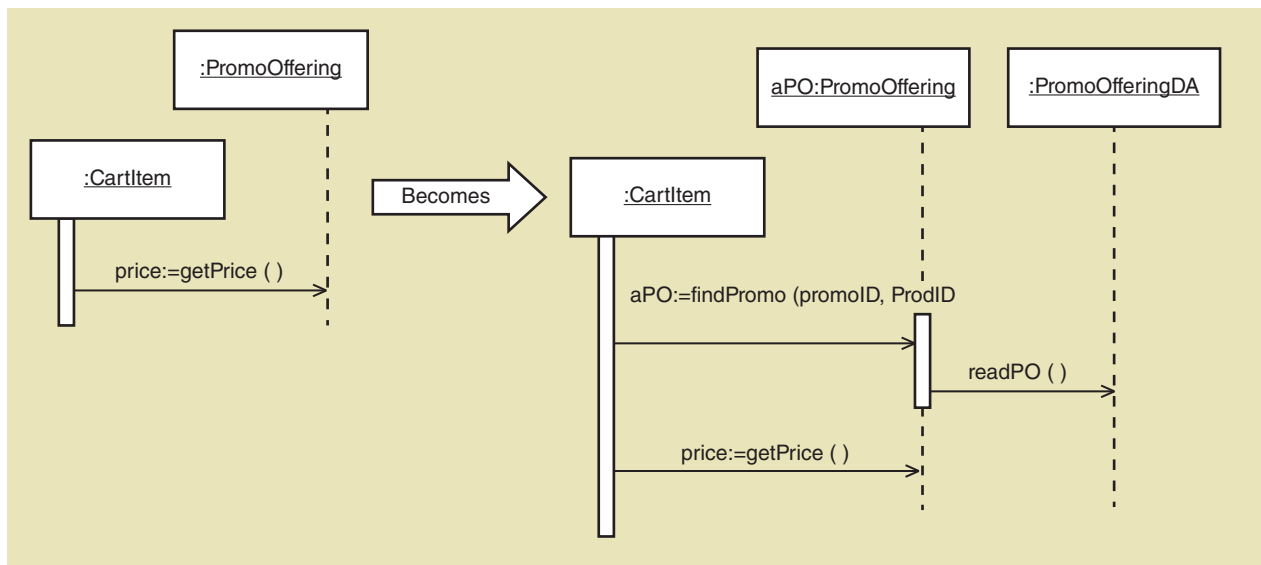


FIGURE 13-21 Partial sequence diagram with data access as part of constructor



When the database needs to be read to retrieve data, usually as part of a constructor, it is a little more complex. There are several techniques (that provide different designs) for linking the domain layer to the data access layer for reading data. There are two common techniques.

One technique is to have the constructor method of an object read the database and obtain the required data to flesh out its attributes and create an object in memory. For example, in the *Fill shopping cart* use case, we need to get the price from a `PromoOffering` object. This means that we first have to have a `PromoOffering` object in memory, so it must be created and then accessed. **Figure 13-21** illustrates this technique. The `findPromo` message invokes a constructor for `:PromoOffering`. During the instantiation, it reads the database to get the required data. Then, it can respond to the `getPrice` message.

Another technique is to send a message to the data access layer object and have it read the database and then instantiate a new problem domain object. This second technique is better when a set of objects needs to be created from a database access that returns an unknown number of rows. However, both techniques are good solutions. **Figure 13-22** illustrates this second technique.

■ The Data Access Layer for the *Fill Shopping Cart* Use Case

Figure 13-23 is a portion of the *Fill shopping cart* use case from Figure 13-16. In this diagram, we have only included the `AddItem` message to limit the complexity of the drawing. The data access classes have all been added along the top. The first two, `:OnlineCartDA` and `:CartItemDA`, are needed so that the system can write out the data from the newly created online cart and cart item. At the bottom of the lifeline of each of those two objects is the `save` statement, which causes the data to be written to the database. The next three data access objects, `:PromoOfferDA`, `:ProductDA`, and `:InventoryDA`, are needed to retrieve data as part of the constructor for `:PromoOffering`, `:ProductItem`, and `:InventoryItem`. Each of those sets of messages function as explained in Figure 13-21. As you can see, adding the data access layer can increase the level of complexity of the drawing quite dramatically.

It is important during this process to ensure that source objects have navigation visibility to destination objects so messages can be sent. We assume but don't show that the data access objects have global visibility. (In your programming class, you will learn that factory or singleton classes are often designed

FIGURE 13-22 Partial sequence diagram with data access prior to constructor

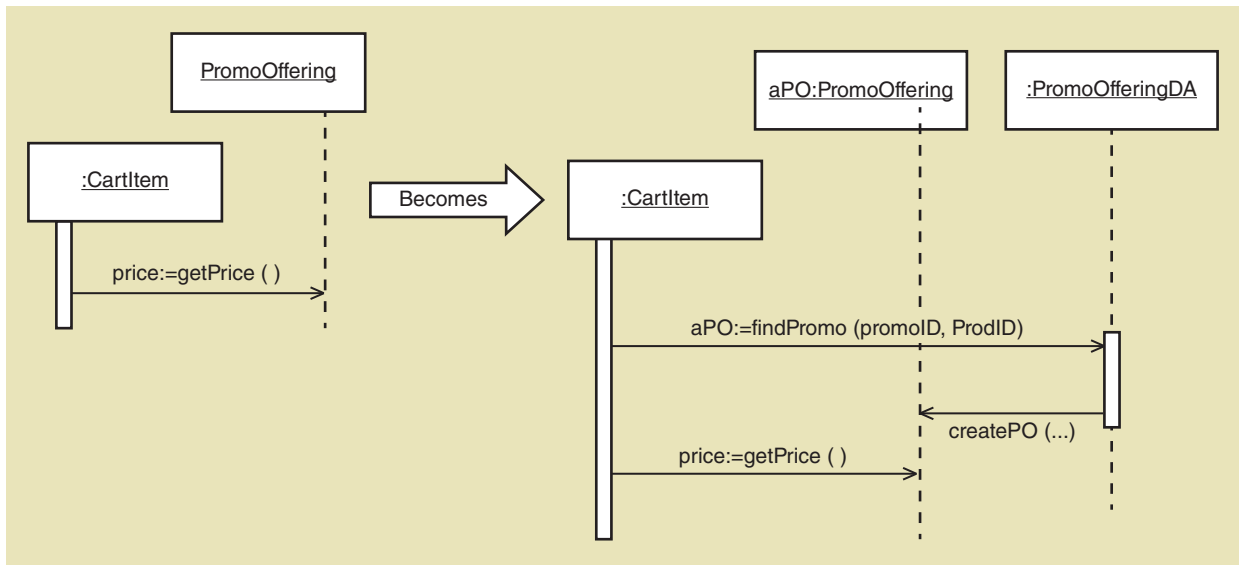
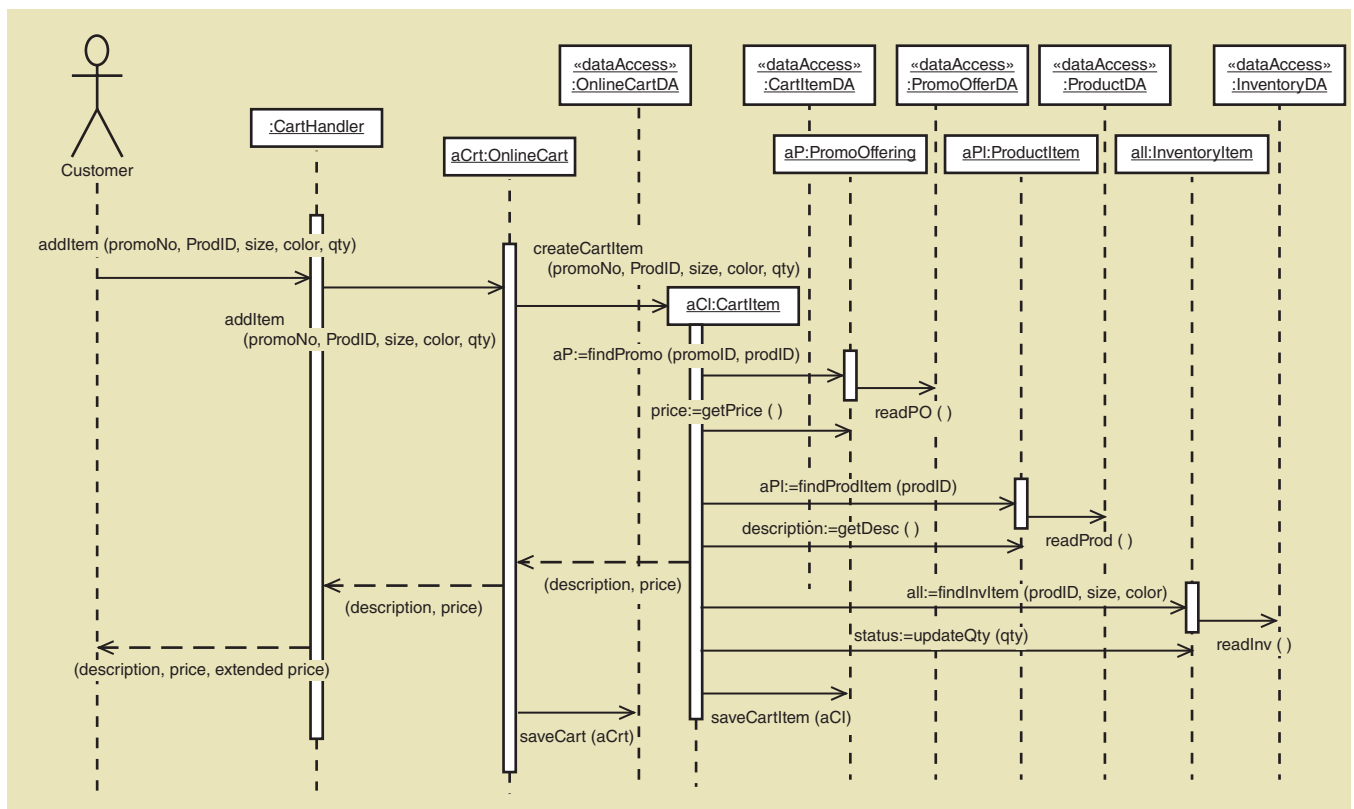


FIGURE 13-23 Partial sequence diagram for the Fill shopping cart use case with data access layer



with global methods.) After the appropriate problem domain object is created, a reference to it is returned to the object that needs visibility. As you look closely at Figure 13-23, note that every object that sends a message to another object must have navigation visibility to that object. Remember this important design point as you develop your design solutions.

An effective method for understanding what is going on in Figure 13-23 is to begin with the internal messages from the sequence diagram in Figure 13-17. Let us review each one and see what changes are required:

- **[firstItem]createCart.** The cart handler is going to send a message to a customer object to create a cart. First, it needs to ensure that there is a customer object in memory. It sends a `findCustomer` message to the `aC:Customer` object to find and create itself from the database. It does so by sending a message to the `:CustomerDA` object to read the database and return the appropriate customer object. Only then can it send the `createCart` message to `aC:Customer`. Also, note that at the end of this execution, the `aCrt:OnlineCart` object sends a message to the data access object to save the data to the database.
- **addItem.** This message is initially the same in both figures. After `aCrt:CartItem` has been created and populated with data, a message is sent to the data access object to save the data to the database.
- **getPrice, getDesc, updateQty.** These three messages all access or update the database. Therefore, each also requires a previous message to find the appropriate data from the database, which is stored in a domain object in memory.

Even though Figure 13-23 appears rather crowded, looking at each internal message to a problem domain class makes the figure easier to understand. The primary thing to remember is that data access objects are necessary to retrieve data and thereby provide navigation visibility to the required object.

■ Updating and Packaging the Design Classes

The design class diagram keeps growing and expanding as each use case is designed and its methods added. The previous examples only created a design class diagram for the problem domain classes. However, design class diagrams can also be developed for each layer. In the view layer and the data access layer, several new classes must be specified. The domain layer also has new classes added for the use case controllers.

As we update the design class diagram, note that there are three types of methods found in most classes: (1) constructor methods, (2) data-get and data-set methods, and (3) use case-specific methods. Constructor methods create new instances of objects. Get and set methods retrieve and update attribute values. To avoid information overload, most developers don't include the get and set methods in the DCD. The third type of method—use case-specific methods—are the ones we normally include in the design class diagram. **Figure 13-24** contains the completed design class diagram for the domain layer classes for the two use cases illustrated in this chapter, *Create customer account* and *Fill shopping cart*.

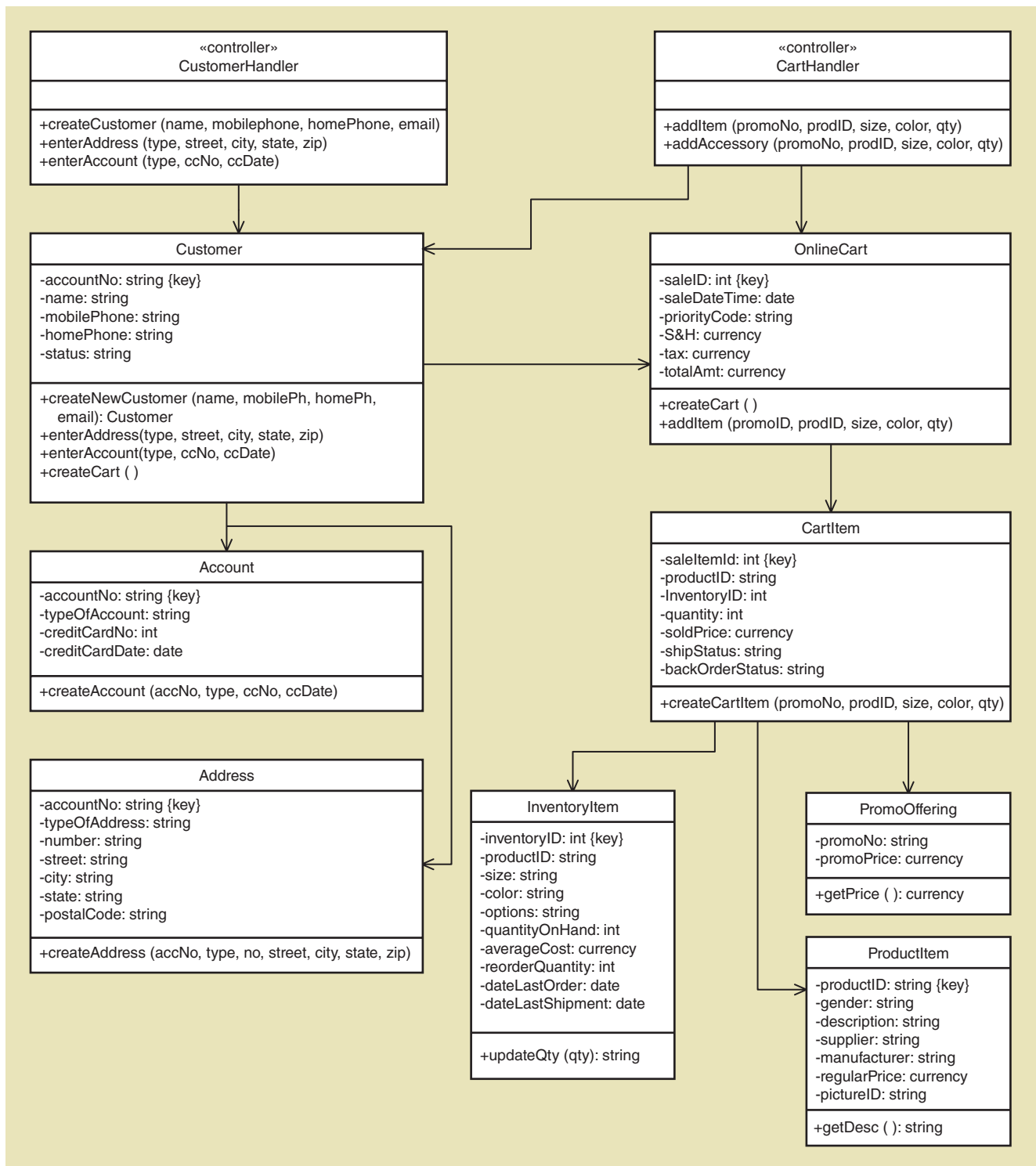
The two major additions to the domain layer classes are the two use case handlers. Additional navigation arrows have also been added to document which classes are visible from the controller classes. The other navigation arrows, which were defined during the first cut of the class diagram, have proved to be adequate for these two use cases.

■ Structuring the Major Components with Package Diagrams

A package diagram in UML is simply a high-level diagram that allows designers to associate classes of related groups. Designers sometimes need to document differences or similarities in relationships in different layers—perhaps separating or grouping objects based on a distributed processing environment. This information can be captured by showing each layer as a separate package. **Figure 13-25** illustrates how these layers might be documented.

The classes are placed inside the appropriate package based on the layer to which they belong. To develop this package diagram, we simply extracted the information

FIGURE 13-24 Updated design class diagram for the domain layer

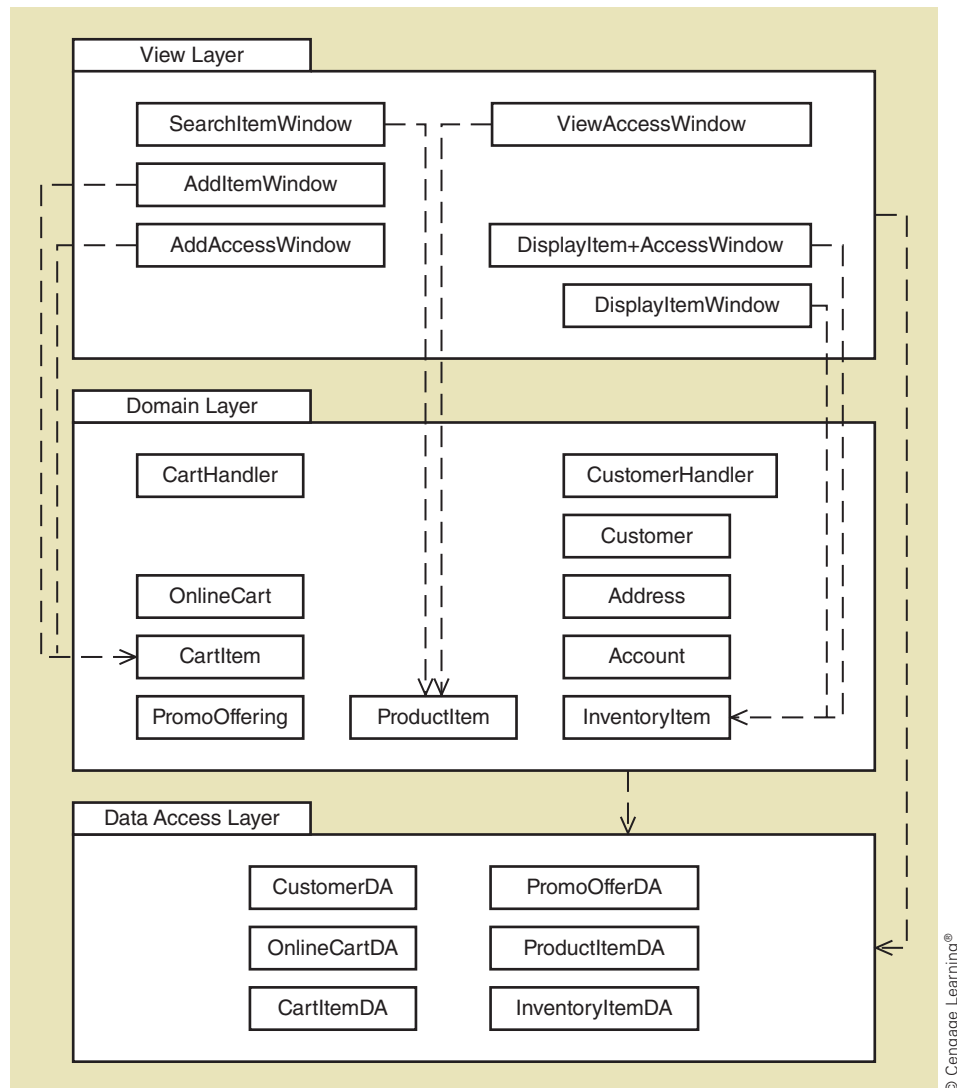


from design class diagrams and interaction diagrams for each use case. Figure 13-25 is only a partial package diagram because the packages contain only the classes from the use case interaction diagrams that were developed in this chapter.

The other symbol used on a package diagram is a dashed arrow, which represents a **dependency relationship**. The arrow's tail is connected to the package that is dependent, and the arrowhead is connected to the independent package. To read a dependency relationship, read it in the direction of the arrow.

dependency relationship a relationship between packages, classes, or use cases in which a change in the independent item requires a change in the dependent item

FIGURE 13-25 Partial design of three-layer package diagram for RMO



For example, SearchItemWindow is dependent on ProductItem. Dependency relationships are used in package diagrams, class diagrams, and even interaction diagrams. A good way to think about a dependency relationship is that if one element changes (the independent element), the other (dependent) element might also have to be changed. Dependency relationships can be between packages or between classes within packages. Figure 13-25 indicates that several classes in the view layer are dependent on classes in the domain layer. Thus, for example, if a change is made in the ProductItem class, the SearchItemWindow class should be evaluated to capture that change. However, the reverse isn't necessarily true. Changes to the view layer usually don't carry through to the domain layer.

■ Implementation Issues for Three-Layer Design

Using design class diagrams, interaction diagrams, and package diagrams, programmers can begin to build the components of a system. Thus, implementation in this sense means constructing the system with a programming language, such as Java, PHP, or such Visual Studio languages as VB or C#. Integrated development environment (IDE) tools have been developed to help programmers construct systems. Such tools as Jbuilder and Eclipse (for Java), Aptana (for PHP), Visual Studio (for Visual Basic), and C# and C++Builder (for C++) provide a high level of programming support, especially in building the view layer classes—the windows and window components of a system.

Unfortunately, these same tools have propagated some bad programming habits in some developers. The ease with which programmers can build GUI windows and automatically insert code has allowed them to put all the code in the windows. Each window component has several associated events where code can be inserted. Thus, some programmers find it easy to build a window with an IDE tool, let the tool automatically generate the class definition, and merely insert business logic code. No new classes need to be defined, and little other coding is required. Some of these tools also have database engines, so the entire system can be built with windows classes. However, taking such shortcuts exacts a price later.

The problem with this approach is the difficulty of maintaining the system. Code snippets scattered throughout the GUI classes are hard to find and maintain. Plus, when the user-interface classes need to be upgraded, the programmer must also find and update the business logic. If a network-based system needs to be enhanced to include a Web front end, a programmer must rebuild nearly the entire system. Or if two user interfaces are desired, all the business logic is programmed twice. Finally, without the tool that generates the code, it is almost impossible to keep the system current. This problem is exacerbated by new releases of the IDE tools, which may not be compatible with earlier versions. Many programmers have had to completely rewrite the front end of a system because the new release of an IDE tool didn't generate code the same way the previous release did. Thus, we advise analysts and programmers to use good design principles in the development of new systems.

Based on the design principle “object responsibility,” it is possible to define which program responsibilities belong to each layer. If you follow these guidelines when writing code, a system will be much easier to maintain throughout its lifetime. Let us summarize the primary responsibilities of each layer.

View layer classes should have programming logic to perform the following:

- Display electronic forms and reports.
- Capture such input events as clicks, rollovers, and key entries.
- Display data fields.
- Accept input data.
- Edit and validate input data.
- Forward input data to the domain layer classes.
- Start and shut down the system.

Domain layer classes should have responsibilities to perform the following:

- Create problem domain (persistent) classes.
- Process all business rules with appropriate logic.
- Prepare persistent classes for storage to the database.

Data access layer classes should have responsibilities to perform the following:

- Establish and maintain connections to the database.
- Contain all SQL statements.
- Process result sets (the results of SQL executions) into appropriate domain objects.
- Disconnect gracefully from the database.

■ Design Patterns

Patterns, also called *templates*, are used repeatedly in everyday life. A chef uses a recipe, which is just another word for a pattern, to combine ingredients into a flavorful dish. A tailor uses a pattern to cut fabric for a great-fitting suit. Engineers take standard components and combine them into established configurations, or set patterns, to build buildings, sound systems, and thousands of other products. Patterns are created to solve problems. Over time and with many

design patterns standard design techniques and templates that are widely recognized as good practice

attempts, people who work on a particular problem develop a given solution to the problem. The solution is general enough that it can be applied over and over again. As time passes, the solution is documented and published, and eventually, it becomes accepted as the standard.

Standard design templates have become popular among software developers because they can speed object-oriented design work. The formal name for these templates is **design patterns**. Design patterns became a widely accepted object-oriented design technique in 1996 with the publication of *Elements of Reusable Object-Oriented Software* by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. These four authors are now referred to as the Gang of Four (GoF). As you learn more about design patterns, you will often see references to a particular design pattern as a GoF pattern. In their book, the authors identified 23 basic design patterns. Today, scores of patterns have been defined—from low-level programming patterns to midlevel architectural patterns to high-level enterprise patterns. Two important enterprise platforms—Java and .NET—have sets of enterprise patterns, which are described in various books and publications.

You are also familiar with the concepts of design patterns and with two specific patterns: three-layer design and use case controllers. Patterns exist at various levels of abstraction. At a concrete level, a pattern may be a class definition that is written in code to be used by any developer. At the most abstract level, a pattern might only be an approach to solving a problem. For example, the multilayer design pattern tends to be more abstract. Thus, multilayer design is an approach to building a system rather than a specific solution.

■ Use Case Controller

In Chapter 12, you were introduced to a design pattern called the Model-View-Controller pattern. Let us formalize the concept of a use case controller and explain its importance as a design pattern. For any particular use case, messages come from the external actor to a windows class (i.e., an electronic input form) and then to a problem domain class. This pattern answers the question of which problem domain class should receive input messages to reduce coupling, maintain highly cohesive domain classes, and maintain independence between the user interface and the domain layer. Designers often define intermediary classes that act as buffers between the user interface and the domain classes.

Figure 13-26 provides a more formal specification for the use case controller pattern. Note that this specification has five main elements:

- Pattern name
- Problem that requires a solution
- Solution to or explanation of the pattern
- Example of the pattern
- Benefits and consequences of the pattern

A use case controller acts as a switchboard, taking input messages and routing them to the correct domain class. In effect, the use case controller acts as an intermediary between the outside world and the internal system. The coupling between the user-interface objects and the problem domain objects is reduced by making a single use case controller object to handle all the input messages. A use case controller also contains logic that controls the flow of execution for the use case.

■ Adapter

The adapter pattern concept is straightforward. The adapter pattern is also a good example of the design principles “protection from variations” and “indirection.” An adapter pattern is roughly akin to an electrical adapter used for international travel. Thus, if you are traveling to England, you might decide to take your hair dryer with you. It has a switch for either 110 volts or 220 volts,

FIGURE 13-26 Pattern specification for the controller pattern

Name:	Controller
Problem:	<p>Domain classes have the responsibility of processing use cases. However, since there can be many domain classes, which one(s) should be responsible for receiving the input messages?</p> <p>User-interface classes become very complex if they have visibility to all of the domain classes. How can the coupling between the user-interface classes and the domain classes be reduced?</p>
Solution:	<p>Assign the responsibility for receiving input messages to a class that receives all input messages and acts as a switchboard to forward them to the correct domain class. There are several ways to implement this solution:</p> <p>(a) Have a single class that represents the entire system, or</p> <p>(b) Have a class for each use case or related group of use cases to act as a use case handler.</p>
Example:	<p>The RMO Customer account subsystem accepts inputs from a <code>:CustomerForm</code> window. These input messages are passed to the <code>:CustomerHandler</code>, which acts as the switchboard to forward the message to the correct problem domain class.</p> <pre> graph TD subgraph UI [User interface] RMO[RMO New Customer] CF[":CustomerForm"] end subgraph DC [Domain classes] CH[":CustomerHandler"] C[":Customer"] end RMO -- createNewCustomer() --> CF CF -- createNewCustomer() --> CH CH -- createNewCustomer() --> C </pre> <p>Other cases of the controller pattern will be used for each RMO use case.</p>
Benefits and consequences:	<p>Coupling between the view layer and the domain layer is reduced. The controller provides a layer of indirection.</p> <p>The controller is closely coupled to many domain classes. If care is not taken, controller classes can become incoherent, with too many unrelated functions.</p> <p>If care is not taken, business logic will be inserted into the controller class.</p>

so you think you can run it on either voltage. However, the plug on the end of the power cord has two flat prongs. Unfortunately, wall sockets in England have slots for three large prongs set at angles. You need something that can adapt the power cord's two prongs to the wall's three angled slots. **Figure 13-27** shows a typical electrical adapter you might use.

FIGURE 13-27 *Electrical adapter*

The adapter design pattern works just like the electrical adapter; it plugs an external component into an existing system. The method signatures on the external class are different from the method names being called from within the system, so the adapter class is inserted to convert the method calls from within the system to the method names in the external class.

Figure 13-28 describes the details of the adapter design pattern. The sample diagram has four UML classes. The one labeled System represents the entire system. The classes within the system use such method names as `getSTax()` and `getUTax()` to access the tax routines. The TaxCalculator class has the method names `findTax1()` and `findTax2()`. The two UML classes in the middle represent the adapter. The top middle class symbol represents an interface class. An interface is useful to specify the method names; although not absolutely necessary, it is a simple way to specify and enforce the use of the correct method names. The adapter class then inherits those method names and provides the method logic for those methods. The body of each method simply extends a call to the final method name `findTax1()` or `findTax2()`. In other words, it “adapts,” or translates, the method names from one to the other.

As you become familiar with this design pattern, you will find that it has a multitude of uses. It is a powerful and elegant solution to making a system more maintainable. Experienced developers use this pattern frequently—for foreign classes and for internally written classes that may need frequent upgrades. It is an excellent way to insulate the system from frequently changing classes.

■ Factory

In the discussions of detailed design, we have often expressed the need to have utility classes, which include the data access objects or controller classes. An adapter in an adapter pattern situation is also a utility class. What class should create these utility objects? In most situations, it doesn’t make sense for domain classes to create them because it isn’t a listed responsibility of domain classes. A popular solution in object-oriented programming is to have some classes that are factories. In other words, these classes instantiate objects from utility classes.

For example, an executing customer object may need to write some data. If the factory class is designed with static methods, which means they have global visibility, the customer object can just say to the factory: “Get me a reference to a data access object for the customer table.” The factory will create a new data access object and return the reference. If a customer data access object already exists in memory, it simply returns the reference. The customer object doesn’t have to be concerned about creating objects to access the database. It just uses whatever is passed to it. This reduces coupling, enhances cohesion, and assigns responsibilities to the right classes. **Figure 13-29** is an example of a factory class.

The factory class has private attributes to hold the references to the data objects that are created. When a request is made to get the reference to a data object, the method simply checks to see if the attribute is null. If so, it creates a new object, places its reference in the attribute, and returns the value. Otherwise, it just returns the parameter with the reference already in it.

FIGURE 13-28 Adapter pattern template

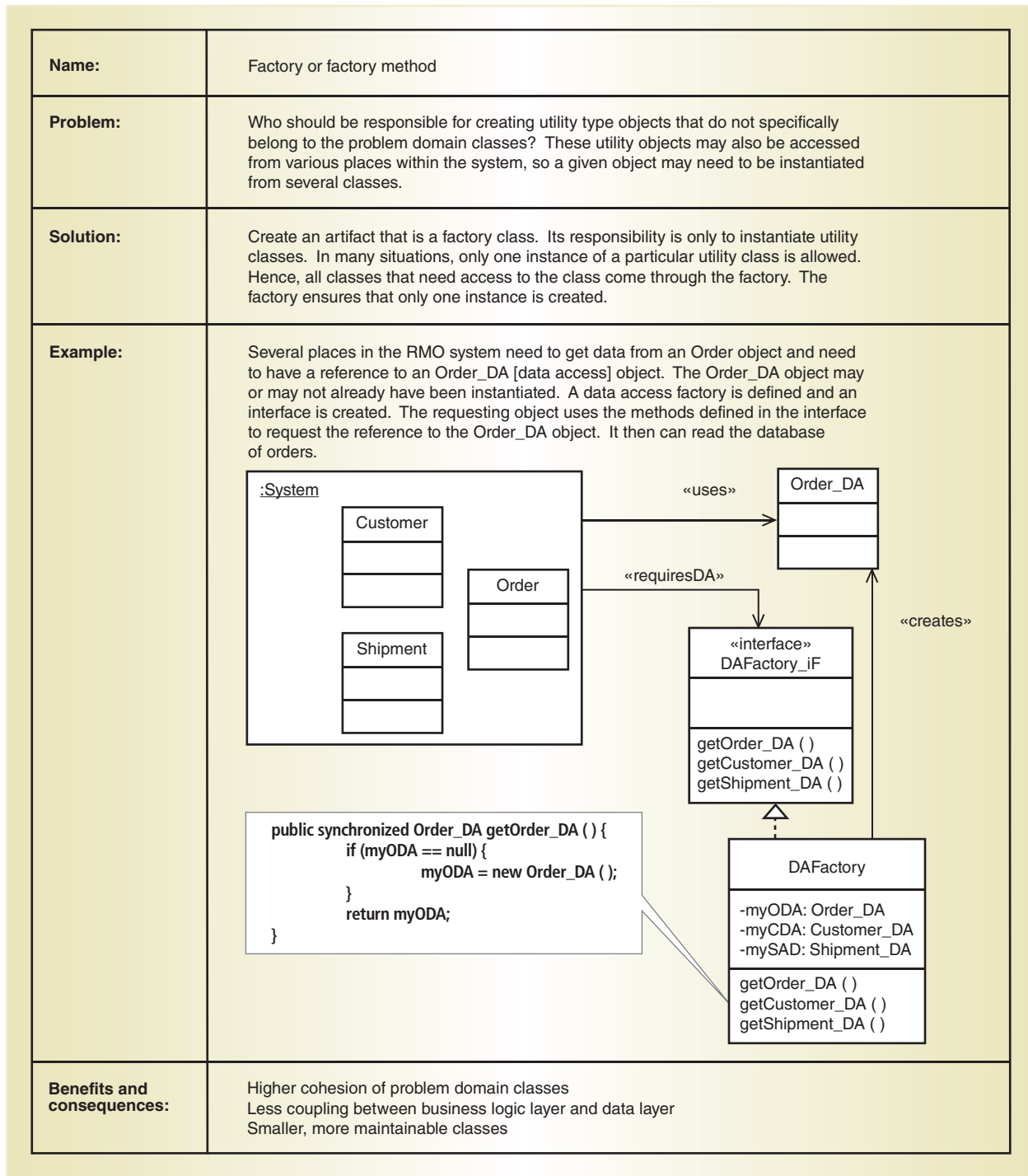
Name:	Adapter
Problem:	A class must be replaced, or is subject to being replaced, by another standard or purchased class. The replacing class already has a predefined set of method signatures that are different from the method signatures of the original class. How do you link in the new class with a minimum of impact so that you don't have to change the names throughout the system to the method names in the new class?
Solution:	Write a new class, the adapter class, which serves as a link between the original system and the class to be replaced. This class has method signatures that are the same as those of the original class (and the same as those expected by the system). Each method then calls the correct desired method in the replacement class with the method signature. In essence, it "adapts" the replacement class so that it looks like the original class.
Example:	<p>There are several places in the RMO system where class libraries were purchased to provide special processing. These purchased libraries provide specialized services such as tax calculations and shipping and postage rates. From time to time, these service libraries are updated with new versions. Sometimes a service library is even replaced with one from an entirely different vendor. The RMO systems staff applies protection from variations and indirection design principles by placing an adapter in front of each replaceable class.</p> <pre> classDiagram class System class TaxCalculatorIF { <<interface>> getSTax () getUTax () } class TaxCalcAdapter { getSTax () getUTax () } class ABCTaxCalculator { findTax1 () findTax2 () } System --> TaxCalculatorIF TaxCalcAdapter .. > TaxCalculatorIF TaxCalcAdapter --> ABCTaxCalculator </pre>
Benefits and consequences:	<p>The adaptee class can be replaced as desired. Changes are confined to the adapter class and do not ripple through the system.</p> <p>Two classes are defined, an interface class and the adapter class.</p> <p>Passed parameters may add more complexity, and it is difficult to limit changes to the adapter class.</p>

■ Singleton

Some classes must have exactly one instance—for example, a factory class or the main window class. Because these classes are instantiated from only one place, it is easy to limit the logic to create only one object.

Other classes must have exactly one instance but can't be easily controlled by having only one place to invoke the constructor. Depending on the system's flow of logic, a particular class might get instantiated from multiple locations. However, only one instance needs to be created, so the first one that needs it

FIGURE 13-29 Factory method pattern template



© Cengage Learning®

creates it, and every other class uses the one that was initially created. Usually, these classes are service classes that manage a system resource, such as a database connection. In fact, the factory class that was just described is an excellent example. This common problem has a standard solution: the singleton pattern.

Figure 13-30 presents the template of the description for the singleton pattern. The singleton pattern provides a solution in which the class itself controls the creation of only one instance.

FIGURE 13-30 Singleton pattern technique

Name:	Singleton
Problem:	Only one instantiation of a class is allowed. The instantiation (new) can be called from several places in the system. The first reference should make a new instance, and later attempts should return a reference to the already instantiated object. How do you define a class so that only one instance is ever created?
Solution:	A singleton class has a static variable that refers to the one instance of itself. All constructors to the class are private and are accessed through a method or methods, such as getInstance(). The getInstance() method checks the variable; if it is null, the constructor is called. If it is not null, then only the reference to the object is returned.
Example:	<p>In RMO's system, the connection to the database is made through a class called Connection. However, for efficiency, we want each desktop system to open and connect to the database only once, and to do so as late as possible. Only one instance of Connection—that is, only one connection to the database—is desired. The Connection class is coded as a singleton. The following coding example is similar to C# and Java:</p> <pre> Class Connection { private static Connection conn = null; public synchronized static getConnection () { if (conn == null) { conn = new Connection (); } return conn; } } </pre> <p>Another example of a singleton pattern is a utilities class that provides services for the system, such as a factory pattern. Because the services are for the entire system, it causes confusion if multiple classes provide the same services.</p> <p>An additional example might be a class that plays audio clips. Since only one audio clip should be played at one time, the audio clip manager will control that. However, for this to work, there must be only one instance of the audio clip manager.</p>
Benefits and consequences:	<p>There are other times when only one instance of an object is needed, but if it is instantiated from only one place, then a singleton may not be required. The singleton object controls itself and ensures that only one instance is created—no matter how many times it is called and wherever the call occurs in the system.</p> <p>The code to implement the singleton is very simple, which is one of the desirable characteristics of a good design pattern.</p>

Notice that the singleton pattern has the same basic logic as the factory method pattern. The difference is that the singleton class has code that applies to itself as static methods. The approach of the singleton solution is that the class has a static variable that refers to the created object. A method such as getConnection is defined and used to get the reference to the object. The first time the getConnection method is called, it instantiates an object and returns a reference to it. On later calls to the method, it simply returns a reference to the already instantiated object. The code is simple and elegant. The example doesn't show the constructor; however, to ensure that only one instance is created, all constructors are specified as private—not accessible—so no other class can accidentally invoke one.

In the singleton template, the pattern is represented by code. To specify this in your design, you should stereotype the class as a «singleton». Good programmers will recognize the stereotype and know exactly how to code the class.

CHAPTER SUMMARY

Detailed design is use case–driven in that each use case is designed separately. This type of design is called *use case realization*. The two primary models used for detailed design are the design class diagram and the sequence diagram.

Detailed design of use cases entails identifying problem domain classes that collaborate to carry out a use case. Each input message from an external actor triggers a set of internal messages. Using a sequence diagram or a communication diagram, the designer identifies and defines all these internal messages. In the first cut, only the problem domain classes and their internal messages are identified. Next, the solution is completed by adding the classes and messages for the view layer and the data access layer.

Three-layer design is part of the overall movement in systems design based on design patterns. A design pattern is a standard solution or template that has proven to be effective to a particular requirement in systems design. The other pattern, introduced in Chapter 12, is a use case controller, which addresses the need to isolate the view layer from the business layer in a simple way that limits coupling between the two layers.

The final step is to convert each message, along with the passed parameters and return values, into method signatures located in the correct classes. This information is used to update the design class diagram. Changes are also made to the design class diagram to show required visibility between the classes in order to send messages in the sequence diagrams. Package diagrams are used to partition the design class diagram into appropriate packages. Dependency between the classes and the packages is also added to the package diagram.

Popular design patterns include the adapter pattern, factory pattern, singleton pattern, and observer pattern. The adapter pattern implements the design principle “protection from variations” by allowing a changing piece of the system to simply plug into a more stable part of the system. When the pluggable piece of the system needs to change, it can just be unplugged and the updated component can be plugged in.

The factory and singleton patterns have much in common. Both return a reference to a specific object. Both allow only one instance of that object to exist in the system. The difference is that the factory pattern enforces a single occurrence for utility classes and the singleton only enforces a single occurrence for itself.

KEY TERMS

activation lifeline

communication diagrams

dependency relationship

design patterns

sequence diagrams

use case realization

REVIEW QUESTIONS

1. What is meant by the term *use case realization*?
2. What are the benefits of knowing and using design patterns?
3. What is the contribution to system development by the Gang of Four?
4. What are the five components of a standard design pattern definition?
5. List five elements included in a sequence diagram.
6. How does a sequence diagram differ from an SSD?
7. What is the difference between designing with CRC cards and designing with interaction diagrams?
8. Explain the syntax of a message on a communication diagram.
9. What is the purpose of the use case controller?
10. What is the purpose of a lifeline on a sequence diagram?
11. What is meant by an activation lifeline? How is it used on a sequence diagram?
12. Describe the three major steps in developing the set of messages for the communication diagram.
13. What assumptions do developers usually make while doing the initial use case realization?
14. When doing multilayer design, what is the order in which layers should be designed? Why?
15. What is the “separation of responsibilities” principle?
16. Explain the two methods of accessing the database to create new objects in memory.

17. What symbols are used in a communication diagram, and what do they mean?
18. Explain the components of message syntax in a sequence diagram. How does this syntax differ from that of a communication diagram message?
19. Explain the method syntax on design classes.
20. What is meant by a dependency relationship? How is it indicated on a drawing?
21. List the major implementation responsibilities of each layer in a three-layer design.
22. What is the purpose of the adapter pattern?
23. What common element is found in the singleton pattern and the factory pattern? What is the basic difference between the two patterns?

PROBLEMS AND EXERCISES

Problems 1 through 7 are based on the solutions you developed in Chapter 5 for problems 1 and 2, which involved a university library system. Alternatively, your instructor may provide you with a use case diagram and a class diagram.

1. **Figure 13-31** is an SSD for the use case *Check out books* in the university library system. Do the following:
 - a. Develop a first-cut sequence diagram that only includes the actor and problem domain classes.
 - b. Develop a design class diagram based on your solution. Be sure to include your controller class.
2. Using your solution to problem 1, do the following:
 - a. Add the view layer classes and the data access classes to your diagram. You may do this with two separate diagrams to make them easier to work with and read.
 - b. Develop a package diagram showing a three-layer solution with view layer, domain layer, and data access layer packages.
3. **Figure 13-32** is an activity diagram for the use case *Return books* in the university library system. Do the following:
 - a. Develop a first-cut sequence diagram that only includes the actor and problem domain classes.
 - b. Develop a design class diagram based on the domain class diagram.
4. Using your solution to problem 3, do the following:
 - a. Add the view layer classes and the data access classes to your diagram.
 - b. Develop a package diagram showing a three-layer solution with view layer, domain layer, and data access layer packages.

FIGURE 13-31 System sequence diagram for the Check out books use case

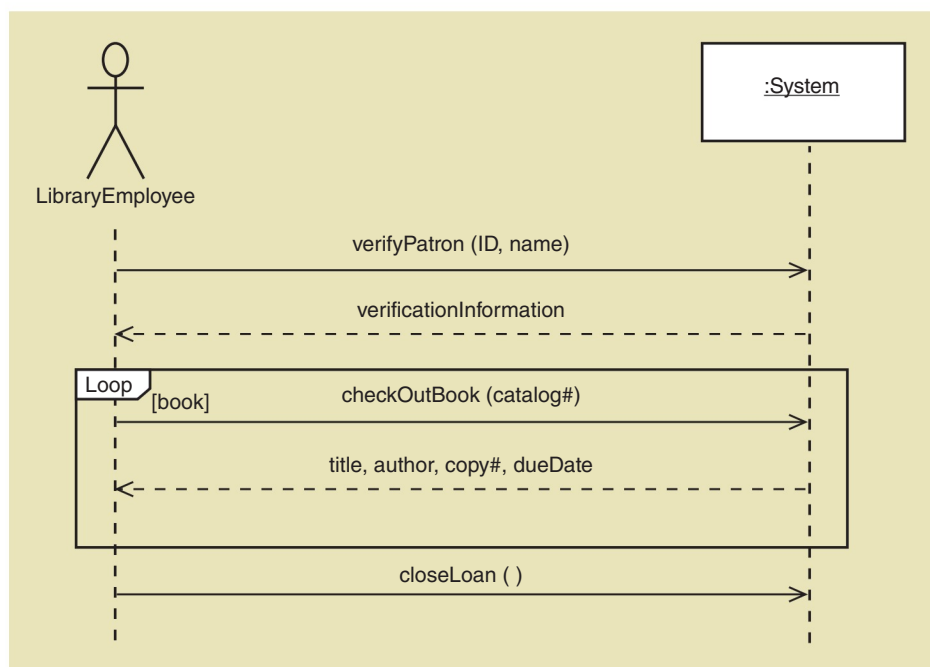
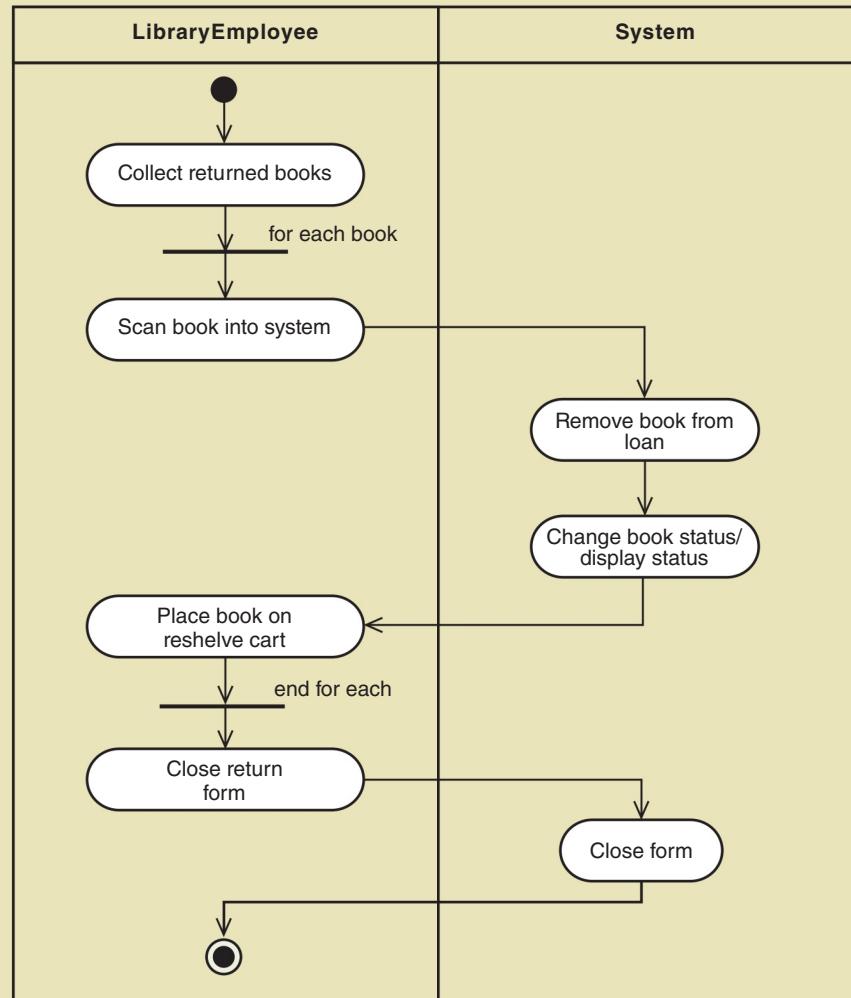


FIGURE 13-32 Activity diagram for the Return books use case



© Cengage Learning®

5. **Figure 13-33** is a fully developed use case description for the use case *Receive new book* in the university library system. Do the following:
 - a. Develop a first-cut communication diagram that only includes the actor and problem domain classes.
 - b. Develop a design class diagram based on the domain class diagram.
6. Using your solution to problem 5, do the following:
 - a. Add the view layer classes and the data access classes to your diagram.
 - b. Develop a package diagram showing a three-layer solution with view layer, domain layer, and data access layer packages.
7. Integrate the design class diagram solutions you developed for problems 1, 3, and 5 into a single design class diagram.

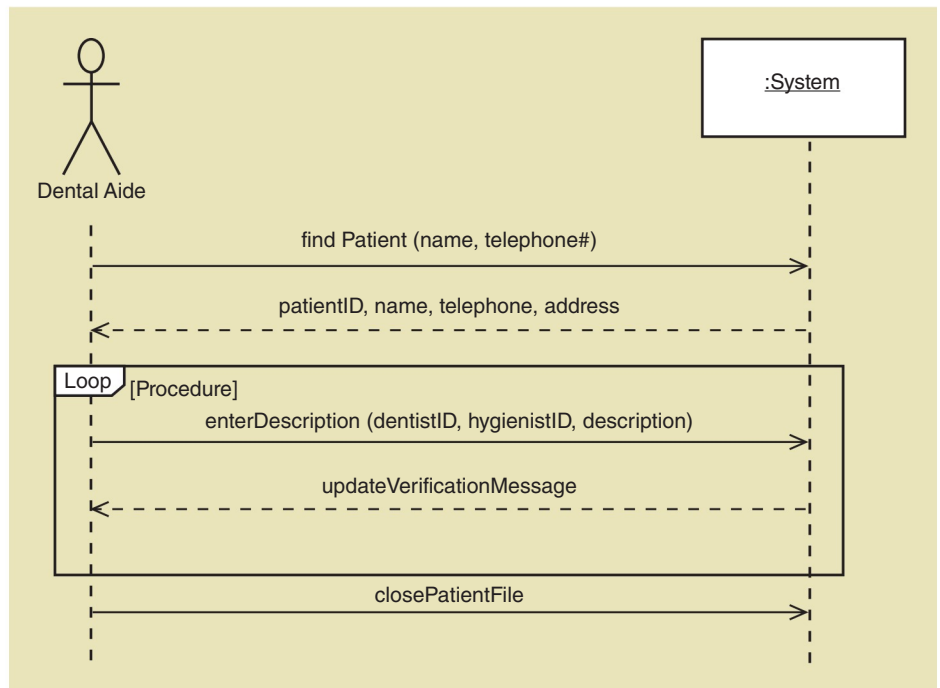
Problems 8 through 14 are based on the solutions you developed for problems 3 and 4 in Chapter 5, which involved a dental clinic system. Alternatively, your instructor may provide you with a use case diagram and a class diagram.
8. **Figure 13-34** is an SSD for the use case *Record dental procedure* in the dental clinic system. Do the following:
 - a. Develop a first-cut sequence diagram that only includes the actor and problem domain classes.
 - b. Develop a design class diagram based on the domain class diagram.
9. Using your solution to problem 8, do the following:
 - a. Add the view layer classes and the data access classes to your diagram.
 - b. Develop a package diagram showing a three-layer solution with view layer, domain layer, and data access layer packages.
10. **Figure 13-35** is an activity diagram for the use case *Enter new patient information* in the dental clinic system. Do the following:
 - a. Develop a first-cut sequence diagram that only includes the actor and problem domain classes.
 - b. Develop a design class diagram based on the domain class diagram.

FIGURE 13-33 Fully developed use case description for the Receive new book use case

Use case name:	Receive new book	
Scenario:	Receive new book	
Triggering event:	Newly purchased book arrives	
Brief description:	The librarian decides on purchases of new books and places order (prior to this use case). Shipments of new books arrive. Each new book is assigned a library catalog number. Some books are simply additional copies of existing titles. Some books are new editions of existing titles. Some books are new titles and new physical books. The new book information is added to the system.	
Actors:	Library Employee	
Stakeholders:	Library Employee, Librarian	
Preconditions:	None	
Postconditions:	Book Title exists, Physical Book exists	
Flow of activities:	Actor	System
	<ol style="list-style-type: none"> 1. Collect new books from receipt of shipment. 2. For each book, research book category and catalog numbers. Assign tentative number. 3a. If new copy of existing title, enter book information and catalog number into system. 3b. If new edition of existing title, enter book information, edition information, and catalog number. 3c. If new title, assign general catalog number. Assign book copy number. 4. Mark book with number. 5. Place book on shelving cart. 6. Repeat for each book (back to step 2). 	<ol style="list-style-type: none"> 3a.1 Update catalog with new number. Verify that not duplicate. 3b.1 Update catalog with new number. Verify that not duplicate. 3c.1 Verify that catalog number not duplicate.
Exception conditions:	Duplicate numbers require further research and reassignment of catalog numbers.	

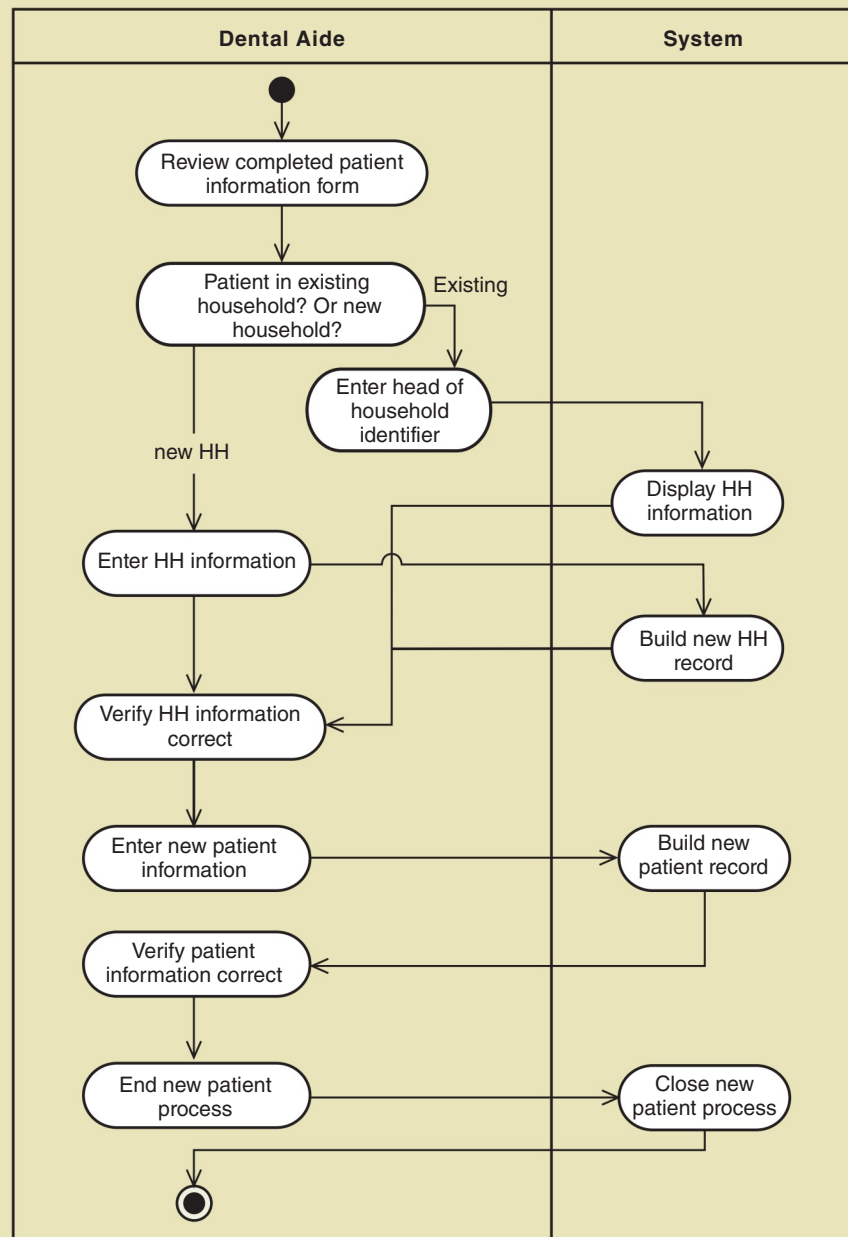
© Cengage Learning®

FIGURE 13-34 System sequence diagram for the Record dental procedure use case



© Cengage Learning®

FIGURE 13-35 Activity diagram for the Enter new patient information use case



© Cengage Learning®

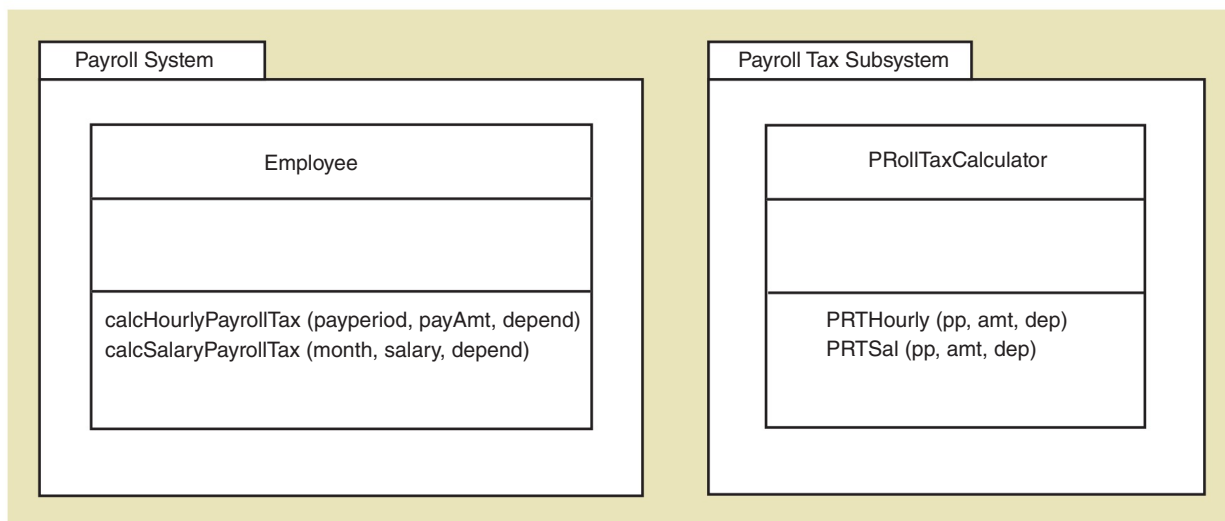
11. Using your solution to problem 10, do the following:
 - a. Add the view layer classes and the data access classes to your diagram.
 - b. Develop a package diagram showing a three-layer solution with view layer, domain layer, and data access layer packages.
12. **Figure 13-36** is a fully developed use case description for the use case *Print patient invoices* in the dental clinic system. Do the following:
 - a. Develop a first-cut communication diagram that only includes the actor and problem domain classes.
 - b. Develop a design class diagram based on the domain class diagram.
13. Using your solution to problem 12, do the following:
 - a. Add the view layer classes and the data access classes to your diagram.
 - b. Develop a package diagram showing a three-layer solution with view layer, domain layer, and data access layer packages.
14. Integrate the design class diagram solutions that you developed for problems 8, 10, and 12 into a single design class diagram.
15. In **Figure 13-37**, the package on the left contains the classes in a payroll system. The package on the right is a payroll tax subsystem. What technique would you use to integrate the payroll tax subsystem into the payroll system? Show how you would solve the problem by modifying the existing classes (in either figure). What new classes would you add? Use UML notation.

FIGURE 13-36 Fully developed use case description for the Print patient invoices use case

Use case name:	Print patient invoices	
Scenario:	Print patient invoices	
Triggering event:	At the end of the month, invoices are printed	
Brief description:	The billing clerk manually checks to see that all procedures have been collected. The clerk spot-checks, using the written records to make sure procedures have been entered by viewing them with the system. The clerk also makes sure all payments have been entered. Finally, he/she prints the invoice reports. An invoice is sent to each patient.	
Actors:	Billing Clerk	
Stakeholders:	Billing Clerk, Dentist	
Preconditions:	Patient Records must exist, Procedures must exist	
Postconditions:	Patient Records are updated with last billing date	
Flow of activities:	Actor	System
	<ol style="list-style-type: none"> 1. Collect all written notes about procedures completed this month. 2. View several patients to verify that procedure information has all been entered. 3. Review log of payments received and verify that payments have been entered. 4. Enter month-end date and request invoices. 5. Verify invoices are correct. 6. Close invoice print process. 	<ol style="list-style-type: none"> 2.1 Display patient information, including procedure records. 3.1 Display patient information, including account balance and last payment transactions. 4.1 Review every patient record. Find unpaid procedures. List on report as aged or current. Calculate and break down by co-pay and insurance pay.
Exception conditions:	None	

© Cengage Learning®

FIGURE 13-37 Payroll system packages and classes



© Cengage Learning®

CASE STUDY

MoveYourBooksNow.com Book Exchange

MoveYourBooksNow.com is a book exchange that does business entirely on the Internet. The company acts as a clearinghouse for buyers and sellers of used books.

To offer books for sale, a person must register with MoveYourBooks. The person must provide a current physical address and telephone number as well as a current e-mail address. The system maintains an open account for this person. Access to the system as a seller is through a secure, authenticated portal.

A seller can list books on the system through a special Internet form. Information required includes all the pertinent information about the book, its category, its general condition, and the asking price. A seller may list as many books as desired. The system maintains an index of all books in the system so buyers can use the search engine to search for books. The search engine allows searches by title, author, category, and keyword.

People who want to buy books come to the site and search for the books they want. When they decide to buy, they must open an account with a credit card to pay for the books. The system maintains all this information on secure servers.

When a request to purchase is made and the payment is sent, TheMoveYourBooks.com sends an e-mail notice to the seller of the book. It also marks the book as sold. The system maintains an open order until it receives notice

that the book has been shipped. After the seller receives notice that a listed book has been sold, the seller must notify the buyer via e-mail within 48 hours. Shipment of the order must be made within 24 hours of the seller sending the notification e-mail. The seller sends a notification to the buyer and TheMoveYourBooks.com when the shipment is made.

After receiving notice of shipment, TheMoveYourBooks.com maintains the order in shipped status. At the end of each month, a check is mailed to each seller for the book orders that have been in shipped status for 30 days. The 30-day waiting period allows the buyer to notify TheMoveYourBooks.com if the shipment doesn't arrive for some reason or if the book isn't in the same condition as advertised.

If they want, buyers can enter a service code for the seller. The service code is an indication of how well the seller is servicing book purchases. Some sellers are very active and use TheMoveYourBooks.com as a major outlet for selling books. Thus, a service code is an important indicator to potential buyers.

For this case, develop the following diagrams:

1. A domain model class diagram
2. A use case diagram
3. SSDs for two use cases, such as *Add a seller* and *Record a book order*
4. A sequence diagram for each of the above use cases
5. An integrated design class diagram that includes classes, methods, and navigation attributes

RUNNING CASE STUDIES

Community Board of Realtors®

In Chapter 3, you identified use cases for the business events for the Community Board of Realtors. In Chapter 5, you elaborated on those use cases. In Chapter 4, you identified the classes associated with the business events. Using your solutions from those chapters, develop:

1. A first-cut DCD by using the problem domain classes that you identified in Chapter 4
2. A communication diagram for the *Create new listing* use case (domain classes and controller class only)
3. A sequence diagram for the *Update agent information* use case (domain classes and controller class only)
4. A multilayer sequence diagram for the *Update agent information* use case that includes domain classes and data access classes
5. A separate multilayer sequence diagram for the *Update agent information* use case that includes the domain classes and the view layer classes
6. A final design class diagram that includes the classes from both use cases; include elaborated attributes, navigation arrows, and all the method signatures from both use cases

The Spring Breaks ‘R’ Us Travel Service

In Chapter 3, you identified use cases for the business events for the Spring Breaks ‘R’ Us Travel Service. In Chapter 5, you elaborated on those use cases. In Chapter 4, you identified the classes associated with the business events. Using your solutions from those chapters, develop:

1. A first-cut DCD by using the problem domain classes you identified in Chapter 4
2. A communication diagram for the *Add a resort* use case (domain classes and controller class only)
3. A sequence diagram for the *Book a reservation* use case (domain classes and controller class only)
4. A multilayer sequence diagram for the *Book a reservation* use case that includes domain classes and data access classes
5. A separate multilayer sequence diagram for the *Book a reservation* use case that includes the domain classes and the view layer classes
6. A final design class diagram that includes the classes from both use cases; include elaborated attributes, navigation arrows, and all the method signatures from both use cases
7. A package diagram of the four subsystems (Resort Relations, Student Booking, Accounting and Finance, and Social Networking) that includes all the problem domain classes

On the Spot Courier Services

In Chapter 10, you developed a first-cut design class diagram and CRC card solutions for two use cases: *Request a package pickup* and *Pickup a package*.

Let us extend your solution from that chapter by developing the following:

1. A sequence diagram for each use case (domain classes and controller classes only).
2. A multilayer sequence diagram for each use case that includes domain classes and data access classes.
3. A separate multilayer sequence diagram for each use case that includes the domain classes and the view layer classes. (We won’t combine view and data access layers on the same drawing. It makes the drawing too complex.)
4. A final design class diagram that includes the classes from both use cases. Include elaborated attributes, navigation arrows, and all the method signatures from both use cases.

In Chapter 9, we defined four subsystems:

- Customer Account subsystem (like Customer Account)
- Pickup Request subsystem (like Sales)
- Package Delivery subsystem (like Order Fulfillment)
- Routing and Scheduling subsystem

Even though these subsystems are somewhat arbitrary, we can treat each one as a separate package. Develop a package diagram for each of the four subsystems by assigning domain model classes to each package. A domain model class should belong to only one subsystem package. Normally, it is the subsystem that instantiates objects from that class. Also, show dependency relationships among the various packages and classes.

Sandia Medical Devices

Review your answers to the case-related questions in Chapter 10 and then do the following:

1. Develop a sequence diagram for the patient use case *View/respond to alert*.
2. Develop a multilayer sequence diagram that includes domain classes and data access classes.
3. Develop a separate multilayer sequence diagram that includes the domain classes and the view layer classes. (We won’t combine view and data access layers on the same drawing. It makes the drawing too complex.)
4. Update your DCD from Chapter 10 to include the methods you have identified. Also, include any changes you may have made to navigation visibility and attribute details.

FURTHER RESOURCES

- Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.
- Grady Booch, et al., *Object-Oriented Analysis and Design with Applications* (3rd ed.). Addison-Wesley, 2007.
- Frank Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley and Sons, 1996.
- Alur Deepak, J. Crupi, and D. Malks, *Core J2EE Patterns: Best Practices and Design Strategies*. Sun Microsystems Press, 2001.
- Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado, *UML 2 Toolkit*. John Wiley and Sons, 2004.
- Erich Gamma, R. Helm, R. Johnson, and J. Vlissides, *Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- Mark Grand, *Patterns in Java*, Volumes I and II. John Wiley and Sons, 1999.
- Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process* (3rd ed.). Prentice Hall, 2004.
- David S. Linthicum, *Next Generation Application Integration: From Simple Information to Web Services*. Addison-Wesley, 2004.
- James Rumbaugh, Ivar Jacobson, and Grady Booch, *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

CHAPTER FOURTEEN

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- ▶ Describe implementation and deployment activities
- ▶ Describe four types of software tests and explain how and why each is used
- ▶ Describe several approaches to data conversion
- ▶ List various approaches to system deployment and describe the advantages and disadvantages
- ▶ Explain the importance of configuration management, change management, and source code control to the implementation, testing, and deployment of a system

CHAPTER OUTLINE

- ▶ Testing
- ▶ Deployment Activities
- ▶ Managing Implementation, Testing, and Deployment
- ▶ Putting It All Together—RMO Revisited

OPENING CASE TRI-STATE HEATING OIL: JUGGLING PRIORITIES TO BEGIN OPERATION

It was 8:30 on Monday morning, and Maria Grasso, Kim Song, Dave Williams, and Rajiv Gupta were about to begin the weekly project status meeting. Tri-State Heating Oil had started developing a new scheduling system for customer orders and service calls five months earlier. The target completion date was 10 weeks away, but the project was behind schedule. Early project iterations had accomplished far less than anticipated because key users had disagreed on what new system requirements to include and the system scope was larger than expected.

Maria began the meeting. “We’ve gained a day or two since our last meeting due to better-than-expected unit testing results,” she said. “All the methods developed last week sailed through unit testing, so we won’t need any time this week to fix errors in that code.”

Kim frowned. “I wouldn’t get too cocky just yet,” she said. “All the nasty surprises in my last project came during integration testing. We’re completing the user-interface classes this week, so we should be able to start integration testing with the business classes sometime next week.”

Nodding enthusiastically, Dave said: “That’s good! We have to finish testing those user-interface classes as quickly as possible because we’re scheduled to start user training in three weeks. I need that time to develop the documentation and training materials and work out the final training schedule with the users.”

Rajiv looked doubtful. “I’m not sure that we should be trying to meet our original training schedule with so much of the system still under development,” he said. “What if integration testing shows major bugs that require more time to fix? And what about the unfinished business and database classes? Can we realistically start training with a system that’s little more than a user interface with half a system behind it?”

“But we have to start training in three weeks,” Dave replied. “We contracted for a dozen temporary workers so we could train our staff on the new system. Half of them are scheduled to start in two weeks

and the rest two weeks after that. It’s too late to renegotiate their start dates. We can extend the time they’ll be here, but delaying their starting date means we’ll be paying for people we aren’t using.”

Maria spoke up. “I think that Rajiv’s concerns are valid,” she said. “It’s not realistic to start training in three weeks with so little of the system completed and tested. We’re at least five weeks behind schedule, and there’s no way we’ll recapture more than four or five days of that during the next few weeks. I’ve already looked into rearranging some of the remaining coding to give priority to the work most critical to user training. There are a few batch processes that can be put on the back burner for a while. Kim, can you rearrange your testing plans to handle all the interactive applications first?”

“I’ll have to go back to my office and take another look at the dependencies among those programs,” Kim replied. “Offhand, I’d say yes, but I need a few hours to make sure.”

“Okay,” Maria said. “Let’s proceed under the assumption that we can rearrange coding and testing to complete a usable system for training in five weeks. I’ll confirm that by e-mail later today, as soon as Kim gets back to me. I’ll also schedule a meeting with the CIO to deliver the bad news about temporary staffing costs.”

After a few moments of silence, Rajiv asked, “So, what else do we need to be thinking about?”

Well, let’s see,” Maria replied. “There’s hardware delivery and setup, operating system and DBMS installation, importing data from the old database, the network upgrade, and stress testing for the distributed database accesses.”

Rajiv smiled and said to Maria, “You must have been a juggler in your youth, which would have been good practice for keeping all these project pieces up in the air. Does management pay you by the ball?”

Maria chuckled. “I do think of myself as a juggler sometimes. And if management paid me by the ball, I could retire as soon as this project is finished!”

■ Overview

Developing any complex system is inherently difficult. For example, consider the complexity of manufacturing automobiles. Tens of thousands of parts must be fabricated or purchased. Laborers and machines must assemble those parts into small subcomponents, such as dashboard instruments, wiring harnesses, and brake assemblies. These are in turn assembled into larger subcomponents, such as instrument clusters, engines, and transmissions, which in turn must be constructed, tested, and passed on to subsequent assembly steps.

The effort, timeliness, cost, and output quality of each step depend on all the preceding steps.

Implementing and deploying an information system is similar in many ways; it is a complex production and assembly process that must use resources efficiently, minimize construction time, and maximize product quality. But unlike automobile manufacturing, it isn't done once and then used to build thousands of similar units. Instead, implementation and deployment are unique to each project and must match that project's characteristics.

We have spent many chapters detailing the first four core processes of the system development life cycle (SDLC). This chapter covers the last two core processes—those related to implementing the system and deploying the system, as shown in **Figure 14-1**.

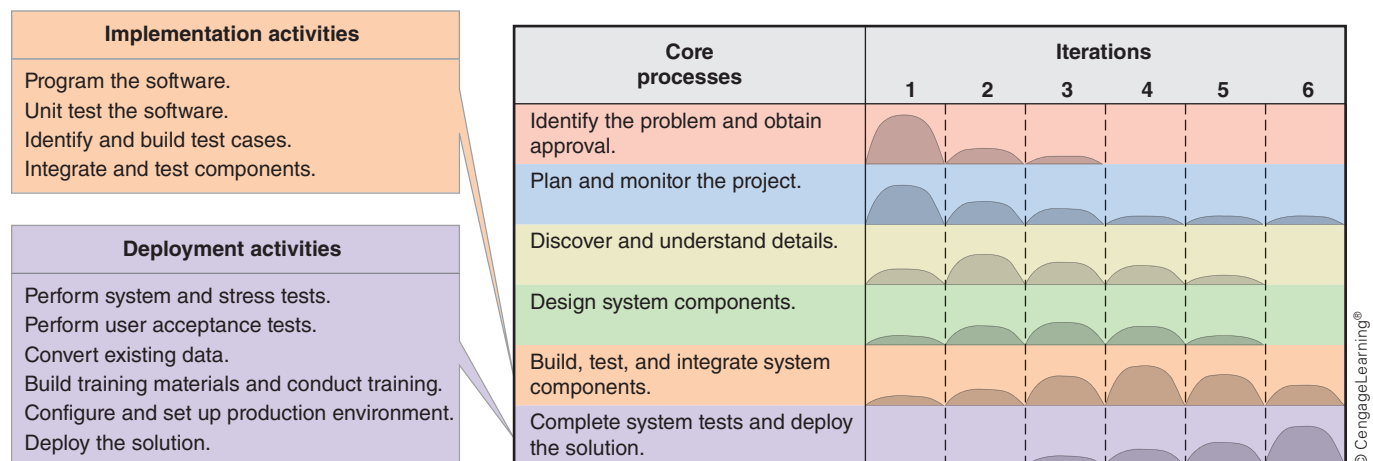
The fact that we are covering two core processes in a single chapter doesn't mean that they are simple or unimportant. Rather, they are complex processes that you will learn about in detail by completing other courses and reading other books as well as through on-the-job training and experience. Our purpose in covering them in this chapter is to round out our discussion of the SDLC and to show how all the core processes and activities relate to one another.

Implementation activities are those related to building and testing the new software, and to integrating all the components together. Earlier, you learned that a new system may include purchased components, prebuilt components, newly developed software components, DBMS components, and other middleware components. All of these components must be tested in the total configuration of the new system. The fifth core process includes activities to complete all these tasks.

Deployment activities are related to final acceptance testing by the users of the system, training the user, converting any existing data to the new DBMS structure, configuring and testing the production computers and network, installing the new system, and turning it on.

Sometimes the dividing line between implementation and deployment is somewhat fuzzy because testing of the new system occurs during both core processes. These two core processes can be distinguished by the primary focus of each. Implementation is focused toward building and testing the system by the technical project team. Deployment is focused toward putting the system into

FIGURE 14-1 Core processes with implementation and deployment activities



production, particularly to ensure the users are satisfied with its capabilities and know how to use the new system. Because testing is such an integrated activity across both core processes, this chapter begins with a discussion of software testing.

■ Testing

Testing activities are a key part of implementation and deployment activities, although different kinds of tests are used in each core process. Testing is the process of examining a component, subsystem, or system to determine its operational characteristics and whether it contains any defects. To conduct a test, developers must have well-defined specifications for both functional and nonfunctional requirements. From the requirements specifications, test developers develop precise definitions of expected operational characteristics. The developers can test software by designing and building the software, exercising its function, and examining the results. If the results indicate a shortcoming or defect, then the project team cycles back through earlier implementation or deployment activities until the shortcoming is remedied or the defect is eliminated.

Test types, their related core processes, and the defects they detect and operational characteristics they measure are summarized in Figure 14-2. Each type of testing is described in detail later in this section.

An important part of developing tests is specifying test cases and data. A **test case** is a formal description of the following:

test case a formal description of a starting state, one or more events to which the software must respond, and the expected response or ending state

- A starting state or condition
- One or more events to which the software must respond
- The expected response or ending state

test data a set of starting states and events used to test a module, group of modules, or entire system

The starting and ending states and the events are represented by a set of **test data**. For example, the starting state of a system may represent a

FIGURE 14-2 Types of software tests and the purpose of each

Test type	Core process	Need and purpose
Unit testing	Implementation	Software components must perform to the defined requirements and specifications when tested in isolation—for example, a component that incorrectly calculates sales tax amounts in different locations is unacceptable.
Integration testing	Implementation	Software components that perform correctly in isolation must also perform correctly when executed in combination with other components. They must communicate correctly with other components in the system. For example a sales tax component that calculates incorrectly when receiving money amounts in foreign currencies is unacceptable .
System and stress testing	Deployment	A system or subsystem must meet both functional and non-functional requirements. For example an item lookup function in a Sales subsystems retrieves data within 2 seconds when running in isolation, but requires 30 seconds when running within the complete system with a live database.
User acceptance testing	Deployment	Software must not only operate correctly, but must also satisfy the business need and meet all user “ease of use” and “completeness” requirements—for example, a commission system that fails to handle special promotions or a data-entry function with a poorly designed sequence of forms is unacceptable.

particular set of data, such as the existence of a particular customer and order for that customer. The event may be represented by a set of input data items, such as a customer account number and order number used to query order status. The expected response may be a described behavior, such as the display of certain information, or a specific state of stored data, such as a canceled order.

Preparing test cases and data is a tedious and time-consuming process. At the component and method levels, every instruction must be executed at least once. Ensuring that all instructions are executed during testing is a complex problem. Fortunately, automated tools based on proven mathematical techniques are available to generate a complete set of test cases. Many test cases representing normal and exceptional processing situations should be prepared for each scenario.

■ Unit Testing

Unit testing is the lowest level of and the earliest testing for a new software system. The concept and approach to unit testing is not consistent across all development teams and, in fact, it has changed over the years as programming methods and languages have changed. In today's object-oriented development approaches, a unit can be defined to be something as small as an individual method. However, a frequent definition is to define a unit as a class or sometimes even a small set of closely integrated classes such as a component. For our purposes, we define **unit test** as a test of an individual method, class, or component before it is integrated with other software. The primary purpose of doing unit testing is to test a small piece of the code in isolation to make sure that it functions correctly before it is integrated into a larger program. Stated another way, its purpose is to make sure that the unit is error-free before being integrated into the larger program. There are three primary characteristics of a unit test:

1. It is done in isolation.
2. The test data and the test are done by the programmer who wrote the code.
3. It is done quickly without a large requirement for other resources.

The first requirement is that the unit test be done in isolation. The objective of a unit test is to test a specific piece of software, and to be able to easily identify the misbehaving code when an error occurs. If the unit is too large, or if it is not isolated, the error could come from some code or an interface other than the specific code being tested. One of the difficulties of being able to isolate the unit being tested is that it often will depend on other software, either to receive a parameter or to output a result. **Figure 14-3** illustrates this condition. Suppose the piece of software is the colored box and the drawing on the left describes the location of the unit within a larger component. As can be seen, the unit to be tested is called from and receives input parameters from another component, and it passes output to two other components. A true unit test would need to be configured as shown in the drawing on the right. A **driver** and a **stub** may need to be written and used to test the unit. In this example, the stubs are overlapped to indicate that a single stub class could be used to receive all unit test results.

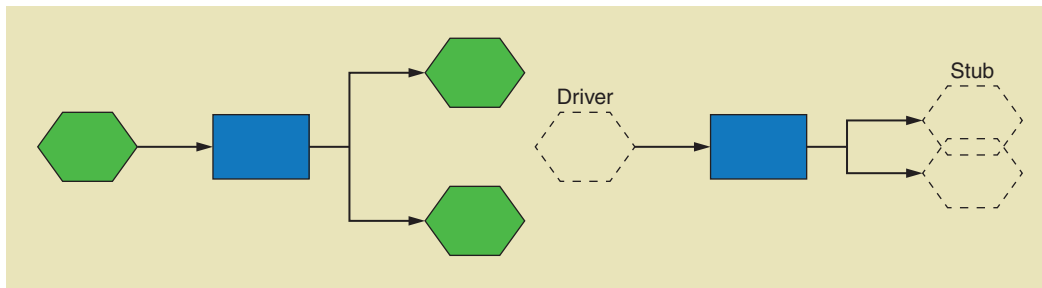
The second requirement is that pieces of code are tested by the programmer who writes the code. This is the fastest and easiest approach to unit testing. The programmer writing the code can quickly generate test data and run several tests on the class or method. In addition, this requirement is in place

unit test test of an individual method, class, or component before it is integrated with other software

driver a method or class used in unit testing that simulates the behavior of a class that calls and sends parameters to the unit being tested

stub a method or class used in unit testing that receives and displays the output from the unit being tested

FIGURE 14-3 Unit testing using drivers and stubs



to place responsibility of writing solid, clean code right on the programmer. It is just human nature that if you are not accountable for your work, the quality degrades substantially. On highly productive development teams, programmers are well trained on how to conduct unit tests. For example, one caveat is that both good test data and bad test data should be used in the unit test. The natural tendency is to just use good test data and observe the expected result. However, a well-trained programmer also thinks of multiple ways that bad data might be received, either from user input or other errors.

Finally, unit testing should not require elaborate test cases or complex testing configurations. Depending on the environment and the language being used, there are also unit test generators that can be used. The test driver and test stub, as shown in the previous figure, can often be written as general purpose and can be used to test many different pieces of code. One of the characteristics of a good unit test is that it can be done quickly and frequently without setting up an entire testing environment. Unit testing should enhance the programmer's productivity, not consume a lot of resources—neither programmer time nor computing resources. Ideally, in an iterative development project where the philosophy is that the system is developed organically, software methods or classes can be written, unit tested, and quickly integrated into the new system as it gradually “grows.”

■ Integration Testing

Integration testing is the next logical extension of unit testing. After small units are tested, they are combined into a larger component and tested together. The objective of integration testing is both to test the interfaces between these units and to test the entire integrated piece of software. An **integration test** evaluates the functional behavior of a group of classes or components when they are combined together. **Figure 14-4** illustrates this concept—a new class or component is integrated into the existing, growing, and tested portion of the system.

The purpose of an integration test is to identify errors that weren't or couldn't be detected by unit testing. There are two types of results that can be tested. The first type of test is to test the interface itself between the components. Such errors may result from a number of problems, including the following:

- **Interface incompatibility.** For example, one method passes a parameter of the wrong data type to another method.
- **Parameter values.** A method is passed or returns a value that was unexpected, such as a negative number for a price.
- **Unexpected state interactions.** The states of two or more objects interact to cause complex failures, as when an OnlineCart class method operates correctly for all possible Customer object states except one.

integration test test of the functional behavior of a group of classes or components when they are combined together

FIGURE 14-4 Integration testing—adding new components to tested components



The second item that is tested is the value of the expected result. Again, both good data and bad data should be used for the test. Good input data should produce the correct expected result. Bad data should be captured and handled appropriately.

Integration testing often begins small, with an individual programmer combining the classes that she has developed into a larger component and tested. However, it soon grows to required components developed by different programmers to be combined and tested together. Consequently, the complexity of integration testing increases rapidly. When integration testing reaches this level, several procedures must be put in place:

- **Building the component for integration test.** Sometimes, particularly when the software is growing organically, this may be a natural result of adding new components. At other times, particular integrated components need to be built and tested separately.
- **Creating test data.** Integration test data is more complex and usually requires coordination between programmers. Responsibility for coordinating the creation, formatting, recording, use, and updating of the test data must be assigned.
- **Conducting the integration test.** Decisions must be made and assignments given about who will conduct the integration tests, how they are done, what resources are used, and how frequently they are executed.
- **Evaluating the results.** Often this requires involvement by all the programmers.
- **Logging test results.** An error log is usually kept at this point to ensure that errors are tracked and corrected. **Figure 14-5** shows a sample error log. There are many commercially available error tracking systems.

FIGURE 14-5 Sample error tracking log

	A	B	C	D	E	F	G	H	I	J
1	Error #	Component	Date test run	Error description	Severity	Priority	Assigned to	Target date	Fixed date	Resolution comment
2	1001	Order entry	11/15/2015	Wrong dollar result	4-Important	4-High	Mary Ann Holmes	11/20/2015	11/20/2105	Equation error
3	1002	Order entry	11/15/2015	System crashed	5-Serious	5-Urgent	Jack Holdaway	11/20/2015	11/19/2015	Passing wrong data type
4										
5										

- **Correcting the code and retesting.** Normally, the programmer who wrote the original code makes the corrections, conducts any required unit tests, and submits the components back for integration testing. The person conducting the integration test usually attempts to identify and isolate the cause of the error as much as possible.

Integration testing of object-oriented software is quite complex. Because an object-oriented program consists of a set of interacting objects that can be created or destroyed during execution, there is often no clear hierarchical structure. As a result, object interactions and control flow are dynamic and complex.

Additional factors that complicate object-oriented integration testing include the following:

- Methods can be (and usually are) called by many other methods, and the calling methods may be distributed across many classes.
- Classes may inherit methods and state variables from other classes.
- The specific method to be called is dynamically determined at run time based on the number and type of message parameters.
- Objects can retain internal variable values (i.e., the object state) between calls. The response to two identical calls may be different due to state changes that result from the first call or occur between calls.

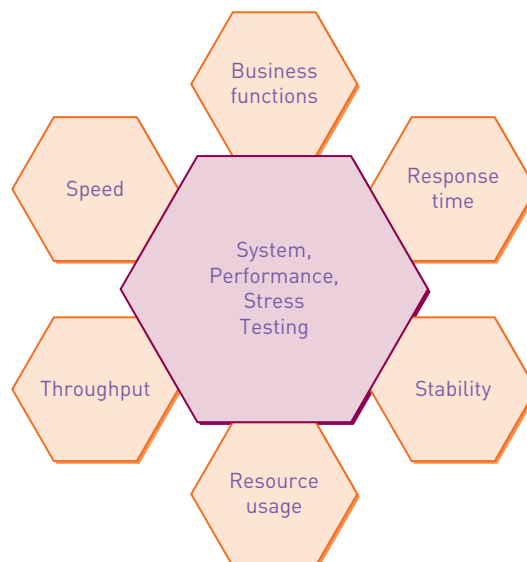
■ System, Performance, and Stress Testing

system test a comprehensive integration test of an entire system or independent subsystem

A **system test** is a comprehensive integration test of the behavior of an entire system or independent subsystem. Integration testing is normally associated with the implementation core process, and system testing is normally associated with the deployment core process. System testing often is used to verify the nonfunctional requirements, such as response time and throughput. The line separating integration testing from system testing is fuzzy, as is the line between implementation and deployment activities. The important differences are scope and timing. Integration tests are performed more frequently and on smaller component groups. System tests are performed less frequently on entire systems or subsystems. In addition, there are various kinds of system tests that test various functional and nonfunctional aspects of the new system. **Figure 14-6** illustrates the types of tests that can be included in system testing.

For a system developed by using a traditional waterfall SDLC, system testing is concentrated near the end of the project. In a typical iterative project,

FIGURE 14-6 Types of tests included in system testing



build and smoke test a system test that is performed daily or several times a week

performance test or **stress test** an integration and usability test that determines whether a system or subsystem can meet time-based performance criteria

response time the desired or maximum allowable time limit for software response to a query or update

throughput the desired or minimum number of queries and transactions that must be processed per minute or hour

user acceptance test (UAT) a system test performed to determine whether the system fulfills user requirements and can support all business and user scenarios

system testing activities are often performed at the end of each iteration where software is completed. Even with systems that are grown organically, there are usually natural plateaus where a subsystem is completed and delivered. System tests are performed on each of these subsystems as they become ready.

System testing may also be performed more frequently. A **build and smoke test** is a system test that is typically performed daily or several times per week. The system is completely compiled and linked (built), and a battery of tests is executed to see whether anything malfunctions in an obvious way (“smokes”).

Build and smoke tests are valuable because they provide rapid feedback on significant integration problems. Any problem that occurs during a build and smoke test must result from software modified or added since the previous test. Daily testing ensures that errors are found quickly and that they can be easily tracked to their sources. Less-frequent testing provides rapidly diminishing benefits because more software has changed and errors are more difficult to track to their sources.

A **performance test**, also called a **stress test**, determines whether a system or subsystem can meet such time-based performance criteria as response time or throughput. **Response time** requirements specify desired or maximum allowable time limits for software responses to queries and updates. **Throughput** requirements specify the desired or minimum number of queries and transactions that must be processed per minute or hour.

Performance tests are complex because they can involve multiple programs, subsystems, computer systems, and network infrastructure. They require a large suite of test data to simulate system operation under normal or maximum load. Diagnosing and correcting performance test failures are also complex. Bottlenecks and underperforming components must first be identified. Corrective actions may include any combination of the following:

- Application software tuning or reimplementation
- Hardware, system software, or network reconfiguration
- Upgrade or replacement of underperforming components

■ User Acceptance Testing (UAT)

A **user acceptance test (UAT)** is comprehensive system testing to determine whether the system fulfills user requirements and can support all business and user scenarios. The UAT is normally the final stage in testing the system. Although the primary focus is usually on the functional requirements, the non-functional requirements are often also verified.

In some cases, UAT is a formal milestone, and requires completion and sign-off by the client. Details of acceptance tests may even be included in a request for proposal (RFP) and procurement contract when a new system is built by or purchased from an external party. In those situations, payments to the developers are often tied to passing specific acceptance tests. In other situations, particularly in Agile projects where the user is involved with the project team, UAT may be less formal and may be integrated into the normal development activities.

In either situation, whether UAT is a formalized process or an informal component of the development, it is a critical component of the development project. All too often, because the project is behind and the delivery date is fast approaching, UAT is minimized or partially skipped. However, minimizing the UAT is always a mistake and a source of problems and disagreements. It is the UAT that verifies that the system is ready to be deployed. If the UAT is minimized or skipped, then it is almost a certainty that the deployed system will have major problems. As mentioned in Chapter 11, the Affordable Care Act insurance Web site is a very visible example. It was deployed with very little UAT, and it took several months before it was ready. It also cost several people their jobs and companies their contracts.

UAT is a big topic, larger than can be covered in this chapter. There are many resources and instruction books on how to plan and execute a successful UAT. You are invited to do further study into this topic. The following sections briefly discuss three areas—planning, preparation, and execution of the UAT.

■ Planning the UAT

The UAT should be included in the total project plan, and whether it will be included in specific iterations or have its own iterations toward the end of the project. Detailed plans for the UAT itself need to be developed early. There are important decisions and preparations that must be done throughout the project. Waiting until late in the project to plan the UAT causes serious difficulties and delays.

In Chapters 2 and 3, you learned about business events, user stories, and use cases. You also learned about functional and nonfunctional requirements. All of these items created requirements and specifications. The UAT should be prepared to test all of them. Test cases should be identified to test various business events, including external events (from the users), temporal events (such as month-end and year-end processing and reporting), and state events (such as inventory order points triggered). You also learned about FURPS+ as a basis for defining specifications. The test plan should address each of these areas and define appropriate test cases to verify the fulfillment of the specification.

The important point here is that as user stories are identified, as use cases are defined, and as nonfunctional requirements are documented, UAT test cases can be identified that will enable the verification of the specifications. In other words, developing the details of what is to be included in the UAT occurs throughout the project. The UAT test plan begins early and continues throughout the project. As each requirement is specified, the following question should be asked: “How can the UAT test that this specification is complete?”

A frequent complaint by developers is that this is a lot of work. The answer is twofold. A simple test identification list can be created and maintained throughout the project. **Figure 14-7** illustrates a possible form. Notice that at this point in the project, it is a fairly simple list that is easily updated as the developers identify requirements. The second part of the answer is, “What is the alternative?” If there is no documentation for what needs to be tested, it is almost guaranteed that the system will have many, and possibly major, defects.

■ Preparation and Pre-UAT Activities

The process of entering data into Figure 14-7 only identifies the potential test cases. The other part of the effort required is to develop the test data. Creating test data can be complex and require substantial resources. There are two primary types of test data: data entered by users and internal data residing in the database. The data entered by users can be precisely defined and documented, or the users can be allowed to create ad hoc data based on the requirement to be

FIGURE 14-7 Simple UAT test case list

	A	B	C	D	E	F
1	Spec ID	Cross refer to use case	Short description	Test conditions	Expected outcomes	Comments
2	10	101	Maintain customer info	Add customer, update customer, delete not allowed	New customer with all fields, updated customer with selected fields	
3	11	201	Maintain sale info	Create sale, update sale, finalize sale, pay for sale	New sale in DB, update selected fields, payment creates transaction	
4	12	202	Ship items	Display items, update status	Sale update, sale items updated, shipment created	

tested. Each has advantages and disadvantages. It is more work to predefine the fields of data, but verification of expected results is easier.

The planning and creation of test data necessitates more detailed planning of the UAT. For example in Figure 14-7, the third specification to ship items can be done with data already existing in the database, or it can use data created by earlier specifications. Hence, the planning of the sequence of tests will assist in the development of the test data. Another example of required planning is the testing of month-end and year-end processing. In both of those cases, there will need to be appropriate detail in the database in order for the test to be meaningful. The question arises whether that data needs to be created from scratch, or if the previous tests will create sufficient data. It should be evident that the creation of test data can be a large endeavor.

Another area of pre-UAT activity is to set up the test environment. Because one of the purposes of the UAT is to verify that the system will function in the day-to-day business environment, it is usually necessary to set up a separate test computer and environment that simulates the true business environment as much as possible. This will include computers, networks, databases, operating systems, and so forth.

■ Management and Execution of the UAT

One of the important decisions is who will participate and who has responsibility for the UAT. Because the objective of the UAT is user acceptance, the system users have primary responsibility. Ideally, system users will take full responsibility for identifying test cases, creating test data, and carrying out the UAT. Unfortunately, in many organizations, the users either are not prepared, do not have enough expertise, or do not have time from their regular responsibilities to take over the testing completely. The user should always have the final say whether a test is successful and a requirement satisfied. However, it is not uncommon that some project personnel are required to help plan, organize, and execute the tests.

Managing the UAT is like a mini-project in and of itself. As mentioned, both users and development personnel are part of the team. Specific tests need to be scheduled. The test data needs to be ready and in place. Specific users will have assignments to complete their tasks. At the conclusion of each test, the results must be verified. If there are failures, the required fixes must be documented and tracked. The error tracking log shown in Figure 14-4 can be used to track the errors. The test case list identified in Figure 14-7 can be expanded to help control and monitor each test and the results of the test. **Figure 14-8** is an example of an expanded version of Figure 14-7 with added columns to help manage the testing process.

FIGURE 14-8 Test case tracking list

	A	B	C	D	E	F	G	H	I	J
1	Spec ID	Cross refer to use case	Short description	Test condition	Expected outcomes	Name of tester	Date executed	Acceptance criteria	Status	Outstanding issues
2	10	101	Maintain customer info	Add customer, update customer, delete not allowed	New customer with all fields, updated customer with selected fields	Mary Helper	7/15/2015	All expected outcomes, DB updated successfully	Accepted	None
3	11	201	Maintain sale info	Create sale, update sale, finalize sale, pay for sale	New sale in DB, update selected fields, payment creates transaction	Mary Helper	7/15/2015	All expected outcomes, DB updated successfully	Pending	1005, 1006
4	12	202	Ship items	Display items, update status	Sale update, sale items updated, shipment created				Not started	

■ Deployment Activities

Once a new system has been developed and tested, it must be placed into operation. Deployment activities (see **Figure 14-9**) involve many conflicting constraints, including cost, the need to maintain positive customer relations, the need to support employees, logistical complexity, and overall risk to the organization. User acceptance and other test types were described in the previous section. Multiple types of tests are often performed concurrently because later project iterations typically include implementation and deployment activities. The following sections provide additional details about deployment activities other than testing.

■ Converting and Initializing Data

An operational system requires a fully populated database to support ongoing processing. For example, online order-entry and management functions of the RMO CSMS rely on stored information about products, promotions, customers, and previous orders. Developers must ensure that such information is present in the database at the moment the subsystem becomes operational.

Data needed at system start-up can be obtained from these sources:

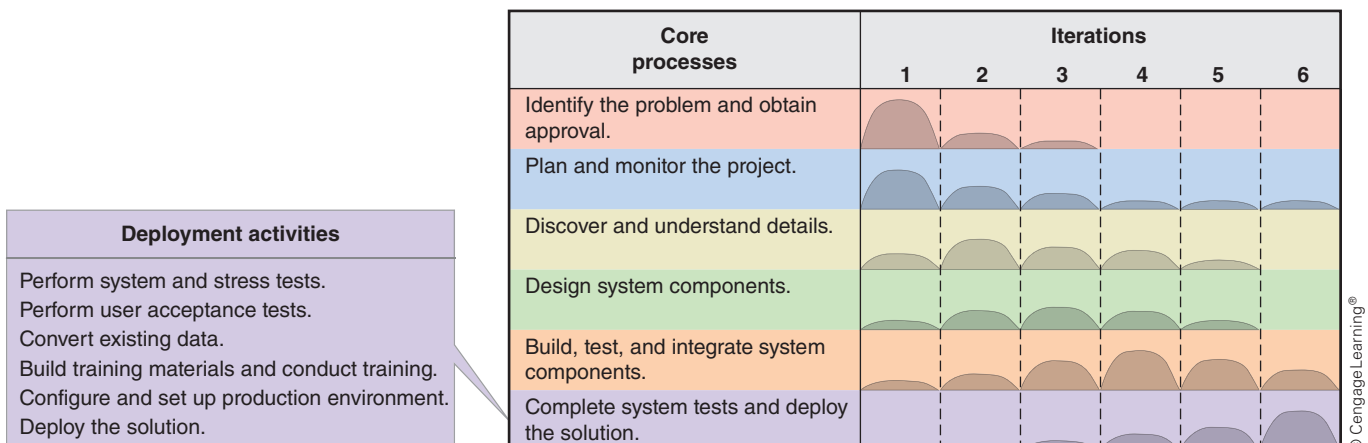
- Files or databases of a system being replaced
- Manual records
- Files or databases from other systems in the organization
- User feedback during normal system operation

■ Reusing Existing Databases

Most new information systems replace or augment an existing manual or automated system. In the simplest form of data conversion, the old system’s database is used directly by the new system with little or no change to the database structure. Reusing an existing database is fairly common because of the difficulty and expense of creating new databases from scratch, especially when a single database often supports multiple information systems, as in today’s enterprise resource planning (ERP) systems.

Although old databases are commonly reused in new or upgraded systems, some changes to database content are usually required. Typical changes include adding new tables, adding new attributes, and modifying existing tables or attributes. Modern database management systems (DBMSs) usually allow database administrators to modify the structure of a fully populated database. Such simple changes as adding new attributes or changing attribute types can be performed entirely by the DBMS.

FIGURE 14-9 SDLC deployment activities



■ Reloading Databases

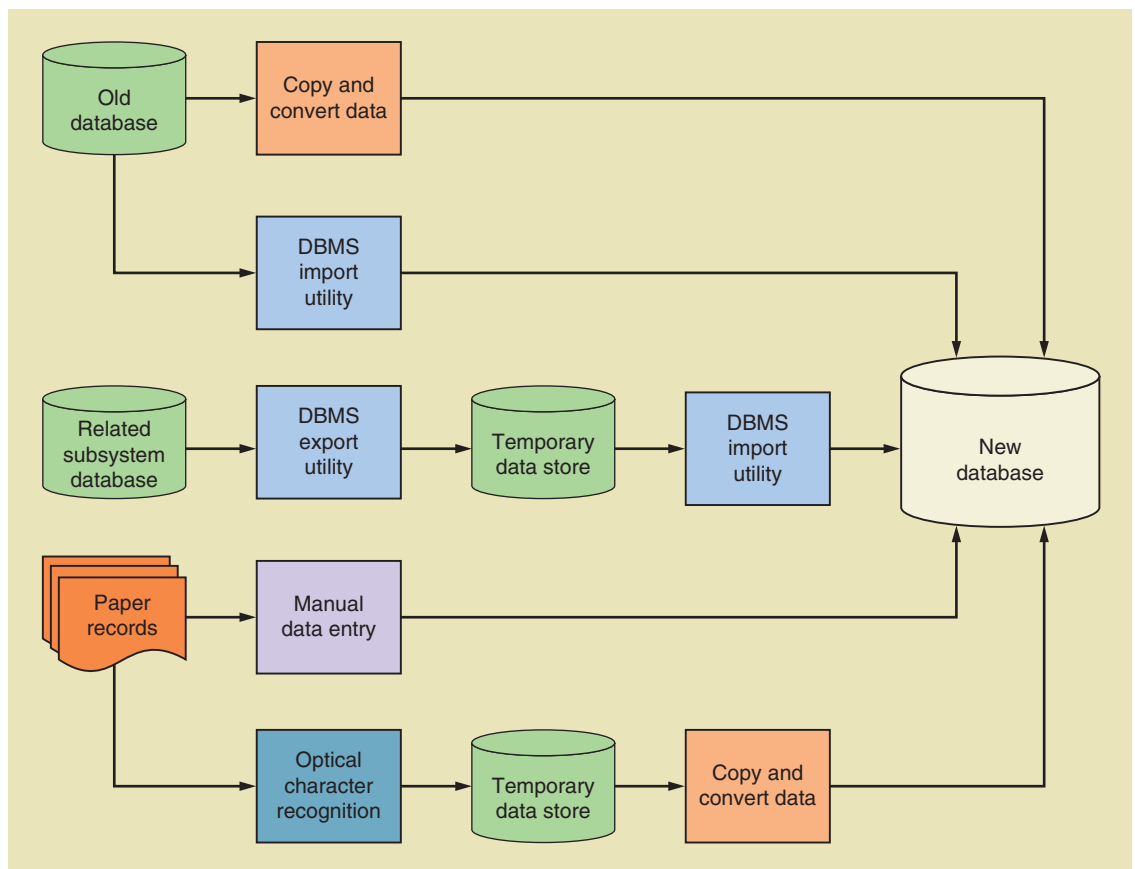
More complex changes to database structure may require creating an entirely new database and copying and converting data from the old database to the new database. Whenever possible, utility programs supplied with the DBMS are used to copy and convert the data. In more complex conversions, implementation staff must develop programs to perform the conversion and transfer some or all of the data. The upper portion of **Figure 14-10** shows both approaches. In either case, the old database can be discarded once the conversion and transfer process is complete.

The middle of **Figure 14-10** shows a more complex approach that uses an export utility, an import utility, and a temporary data store. This approach might be employed when the source and target databases employ different database technologies; no utility exists that can directly translate from one to the other, but a “neutral” format exists that can serve as a bridge.

Data from paper records can be entered by using the same programs being developed for the operational system. In that case, data-entry programs are usually developed and tested as early as possible. Initial data entry can be structured as a user training exercise. For greater efficiency, data from paper records can also be scanned into an optical character recognition program and then entered into the database by using custom-developed conversion programs or a DBMS import utility.

In some cases, it may be possible to begin system operation with a partially or completely empty database. For example, a customer order-entry system need not have existing customer information loaded into the database. Customer information could be added the first time a customer places an order, based on a dialogue between a telephone order-entry clerk and the customer. Adding data as they are encountered reduces the complexity of data conversion but at the expense of slower processing of initial transactions.

FIGURE 14-10 Complex data-conversion example



■ Training Users

Training two classes of users—end users and system operators—is an essential part of any system deployment project. End users are people who use the system from day to day to achieve the system’s business purpose. System operators are people who perform administrative functions and routine maintenance to keep the system operating. **Figure 14-11** shows representative activities for each role. In smaller systems, a single person may fill both roles. In larger organizations, the technical support staff may already be prepared to handle the system operator responsibilities. In that case, only some documentation or basic instructions may be all that is needed.

The nature of training varies with the target audience. Training for end users must emphasize hands-on use for specific business processes or functions, such as order entry, inventory control, or accounting. If the users aren’t already familiar with those procedures, training must include them. Widely varying skill and experience levels call for at least some hands-on training, including practice exercises, questions and answers, and one-on-one tutorials. Self-paced training materials can fill some of this need, but complex systems also require some face-to-face training. If there is a large number of end users, group training sessions can be used, and a subset of well-qualified end users can be trained and then pass their knowledge on to other users.

Determining the best time to begin formal training can be difficult. On one hand, users can be trained as parts of the system are developed and tested, which ensures that they hit the ground running. On the other hand, starting early can be frustrating to users and trainers because the system may not be stable or complete. End users can quickly become frustrated when using a buggy, crash-prone system with features and interfaces that are constantly changing.

In an ideal world, training doesn’t begin until the interfaces are finalized and a test version has been installed and fully debugged. But the typical end-of-project crunch makes that approach a luxury that is often sacrificed. Instead, training materials are normally developed as soon as the interfaces are reasonably stable, and end-user training begins as soon as possible thereafter. It is much easier to provide training if system interfaces are completed in early project iterations.

Documentation and other training materials are usually developed before formal user training begins. Each documentation type is targeted to a different purpose and audience. Documentation can be loosely classified into two types:

system documentation descriptions of system requirements, architecture, and construction details, as used by maintenance personnel and future developers

user documentation descriptions of how to interact with and use the system, as used by end users and system operators

- **System documentation.** Descriptions of system requirements, architecture, and construction details
- **User documentation.** Descriptions of how to interact with and use the system

■ System Documentation

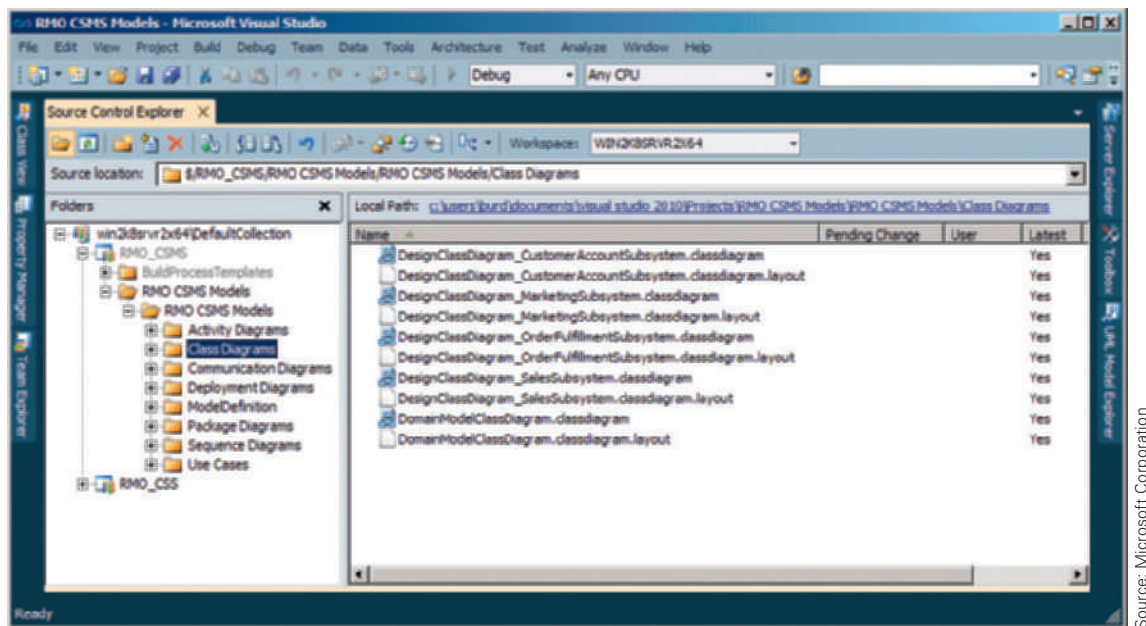
System documentation serves one primary purpose: providing information to developers and other technical personnel who will build, maintain, and upgrade the system. System documentation is generated throughout the SDLC by each

FIGURE 14-11 Typical activities of end users and system operators

End-user activities	System operator activities
Creating records or transactions	Starting or stopping the system
Modifying database contents	Querying system status
Generating reports	Backing up data to archive
Querying database	Recovering data from archive
Importing or exporting data	Installing or upgrading software

© Cengage Learning®

FIGURE 14-12 System documentation stored within Microsoft Visual Studio



Source: Microsoft Corporation

core process and many development activities. System documentation developed during early project iterations guides activities in later iterations, and documentation developed throughout the SDLC guides future system maintenance and upgrades.

A system deployed for a customer is a collection of computing and network hardware, system software, and application software. Once the system had been developed, separate descriptions of it, such as written text and graphical models, are redundant with the system itself. In the early days of computing, there were few automated tools to support development of analysis and design models and even less support for automating the process of generating application software from those models. Developers in that era faced a recurring dilemma: how to minimize the duplicate effort of updating models and application software while ensuring that the system documentation was always “in sync” with the actual system. In the rush to complete and deploy systems and to maintain and upgrade them over time, system documentation updates were often neglected and documentation was frequently lost.

Modern application development tools and methods have largely solved the system documentation dilemma of earlier times. A modern integrated development environment provides automated tools to support all SDLC core processes. Requirements and design models, such as use case descriptions, class diagrams, and sequence diagrams, are developed by using the development tool and stored in a project library (see [Figure 14-12](#)). Changes to one model are automatically synchronized with related models. Application software is often generated in part or in its entirety directly from design models. When application software is altered at a later date, the development tools can “reverse engineer” appropriate changes to the models. Due to these capabilities, system documentation is always complete and in sync with the deployed system, thus simplifying future maintenance and upgrades.

■ User Documentation

User documentation provides ongoing support for end users of the system. It primarily describes routine operation of the system, including such functions as

data entry, output generation, and periodic maintenance. Topics typically covered include the following:

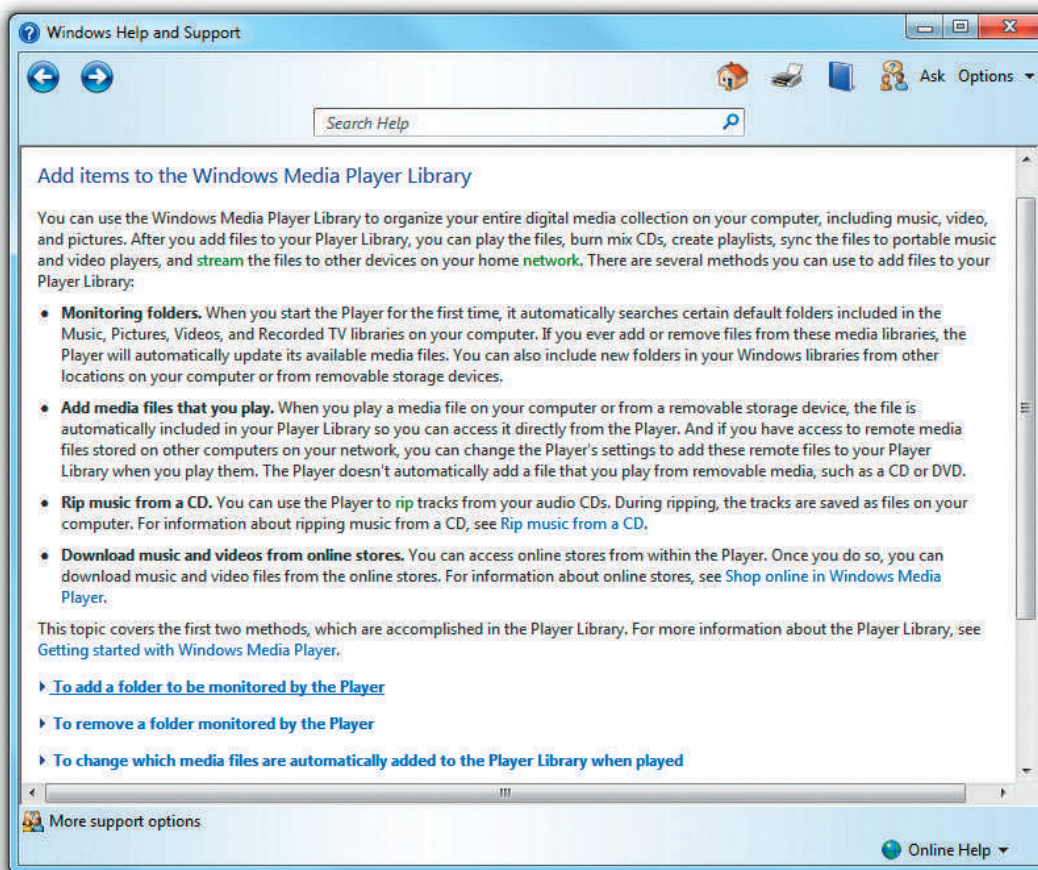
- Software start-up and shutdown
- Keystroke, mouse, or command sequences required to perform specific functions
- Program functions required to implement specific business procedures (e.g., the steps followed to enter a new customer order)
- Common errors and ways to correct them

For ease of use, user documentation typically includes a table of contents, a general description of the purpose and function of the program or system, a glossary, and an index.

User documentation for modern systems is almost always electronic and is usually an integral part of the application. Most modern operating systems provide standard facilities to support embedded documentation. **Figure 14-13** shows electronic user documentation of a typical Windows application. The table of contents can be displayed by clicking the book-shaped icon in the top toolbar, and the user can search for specific words or phrases by using the search tool. The center portion of the display shows individual pages of user documentation. The sample page includes embedded glossary definition hyperlinks and hyperlinks to other documentation pages.

Developing good user documentation requires special skills and considerable time and resources. Writing clearly and concisely, developing effective presentation graphics, organizing information for easy learning and access, and communicating effectively with a nontechnical audience are skills for which

FIGURE 14-13 Sample Windows Help and Support display



Source: Microsoft Corporation

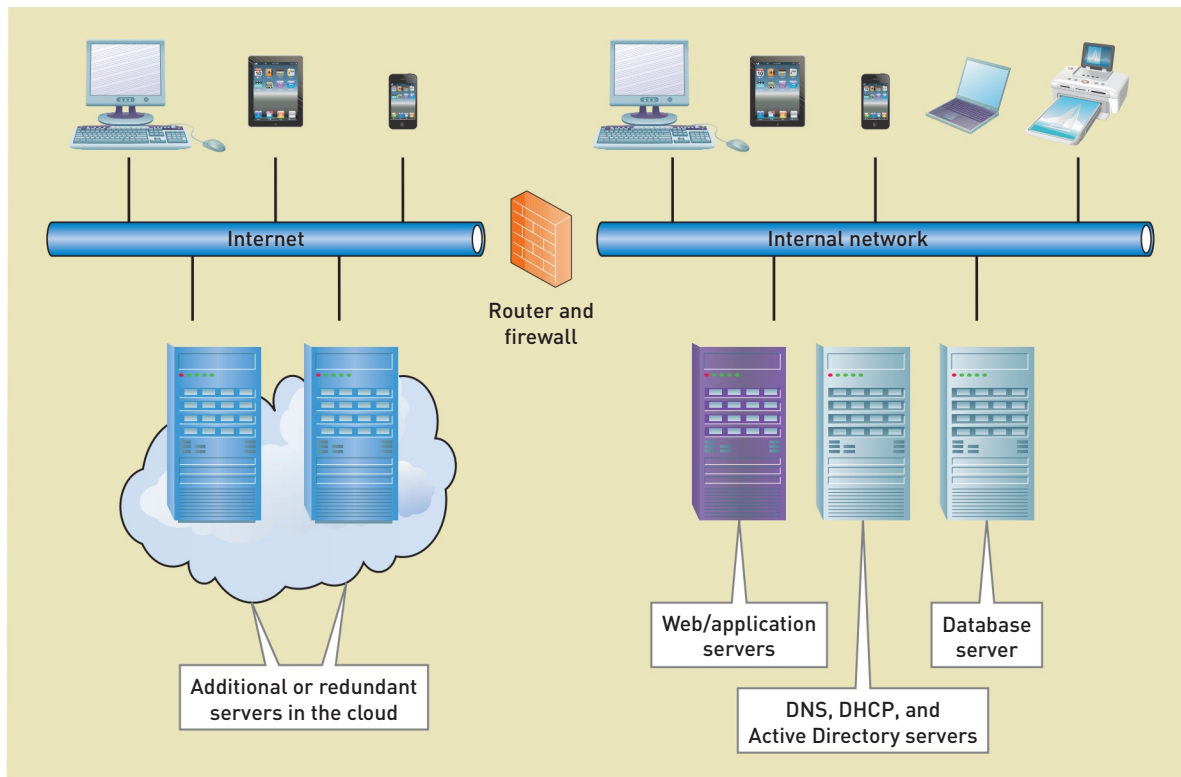
there is high demand and limited supply. Development takes time, and high-quality results are achieved only with thorough review and testing. Unfortunately, preparing user documentation is often left to technicians lacking in one or more necessary skills. Also, preparation time, review, and testing are often shortchanged because of schedule overruns and the last-minute rush to tie up all the loose ends of implementation.

■ Configuring the Production Environment

Modern applications are built from software components based on interaction standards, such as Common Object Request Broker Architecture (CORBA), Simple Object Access Protocol (SOAP), and Java Platform Enterprise Edition (Java EE). Each standard defines specific ways in which components locate and communicate with one another. Each standard also defines a set of supporting system software to provide needed services, such as maintaining component directories, enforcing security requirements, and encoding and decoding messages across networks and other transport protocols. The exact system software, its hardware, and its configuration requirements vary substantially among the component interaction standards.

Figure 14-14 shows a typical support infrastructure for an application deployed using Microsoft .NET, a variant of SOAP. Application software components written in such programming languages as Visual Basic and C# are stored on one or more application servers. Other required services include a Web server for browser-based interfaces, a database server to manage the database, an Active Directory server to authenticate users and authorize access to information and software resources, a router and firewall, and a server to operate such low-level Internet services as domain naming system (DNS) and Internet address allocation (DHCP).

FIGURE 14-14 Infrastructure and clients of a typical .NET application



Unless it already exists, all this hardware and system software infrastructure must be acquired, installed, and configured before application software can be installed and tested. In most cases, some or all of the infrastructure will already exist—to support existing information systems. In that case, developers work closely with personnel who administer the existing infrastructure to plan the support for the new system. In either case, this deployment activity typically starts early in the project so software components can be developed, tested, and deployed as they are developed in later project iterations.

■ Managing Implementation, Testing, and Deployment

The previous sections have discussed the implementation, testing, and deployment activities in isolation. This section concentrates on issues that impact all those activities as well other core processes, including project planning and monitoring, analysis, and design. In an iterative development project, activities from all core processes are integrated into each iteration and the system is analyzed, designed, implemented, and deployed incrementally. But how does the project manager decide which portions of the system will be worked in early iterations and which in later iterations? And how does he or she manage the complexity of so many models, components, and tests?

Some of these issues were partly addressed in earlier chapters. But now that you understand implementation, testing, and deployment activities in depth, you can see that there are many interdependencies that must be accounted for. These interdependencies must be fully identified and considered when developing a workable iteration plan. Furthermore, automated tools must be utilized to manage each part of the development project and to ensure maximal coordination across iterations, core processes, and activities.

■ Development Order

One of the most basic decisions to be made about developing a system is the order in which software components will be built or acquired, tested, and deployed. Choosing which portions of the system to implement in which iterations is difficult, and developers must consider many factors, only some of which arise from the software requirements. Some of the other factors discussed in earlier chapters include the need to validate requirements and design decisions and the need to minimize project risk by resolving technical and other risks as early as possible.

A development order can be based directly on the structure of the system itself and its related issues, such as use cases, testing, and efficient use of development staff. Several orders are possible, including the following:

- Input, process, output
- Top-down
- Bottom-up
- Use-case driven

Each project must adapt one or a combination of these approaches to specific project requirements and constraints.

■ Input, Process, Output Development Order

The **input, process, output (IPO) development order** is based on data flow through a system or program. Programs or modules that obtain external input are developed first. Programs or modules that process the input (i.e., transform

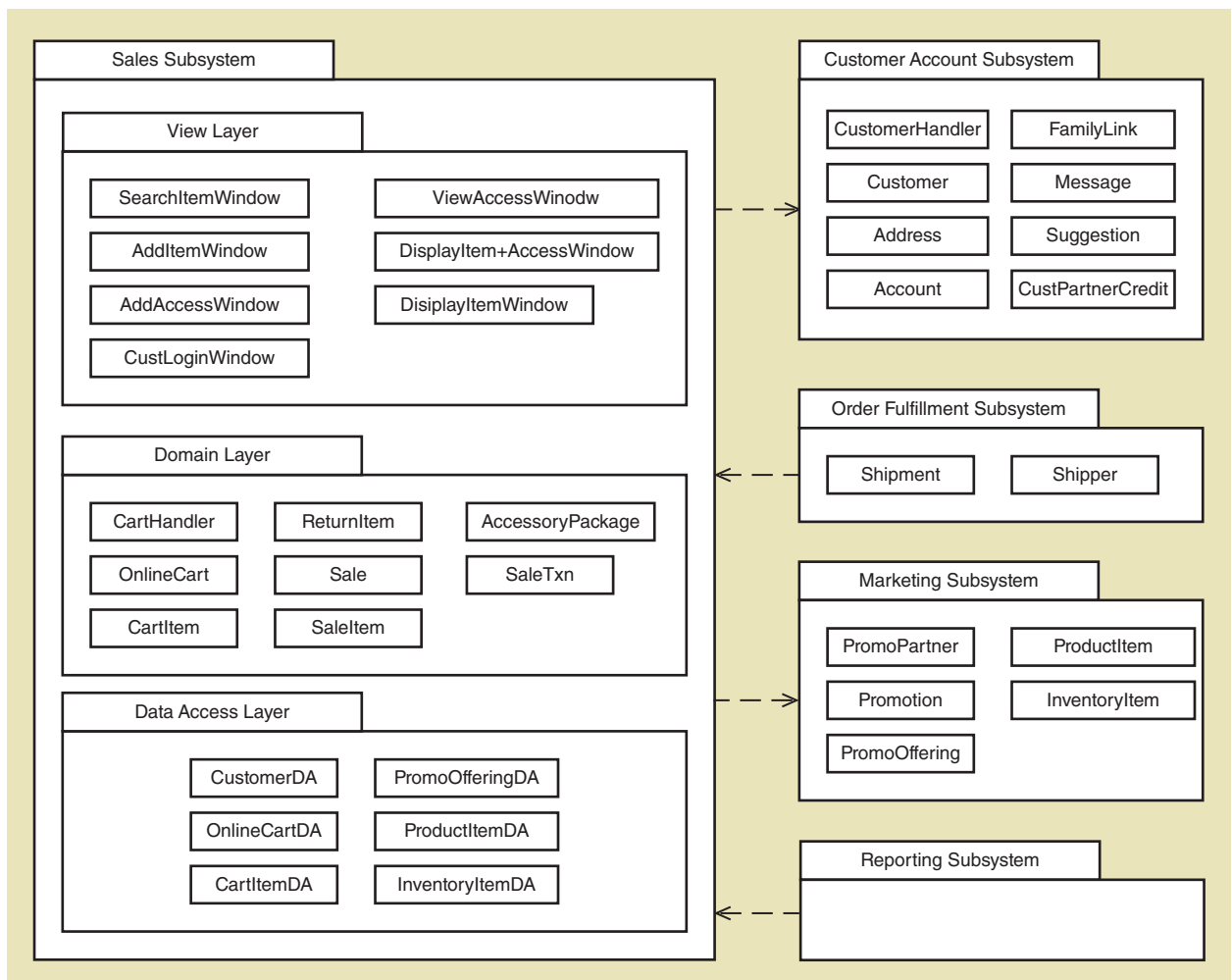
input, process, output (IPO) development order a development order that implements input modules first, process modules next, and output modules last

it into output) are developed next. Programs or classes that produce output are developed last. The key issue to analyze is dependency—that is, which classes and methods capture or generate data that are needed by other classes or methods? Dependency information is documented in package diagrams and may also be documented in a class diagram. Thus, either or both diagram types can guide implementation order decisions.

For example, the package diagram in **Figure 14-15** shows that the Customer Account and Marketing subsystems don't depend on any of the other subsystems. The Sales subsystem depends on the Customer Account and Marketing subsystems, and the Order Fulfillment and Reporting subsystems depend on the Sales subsystem.

The chief advantage of the IPO development order is that it simplifies testing. Because input programs and modules are developed first, they can be used to enter test data for process and output programs and modules. IPO development order is also advantageous because important user interfaces (e.g., data-entry routines) are developed early. User interfaces are more likely to require change during development than during other portions of the system, so early development allows for early testing and user evaluation. If changes are needed, there is still plenty of time to make them. Early development of user interfaces also provides a head start for related activities, such as training users and writing documentation.

FIGURE 14-15 Package diagram for the four RMO subsystems



■ Top-Down and Bottom-Up Development Order

The terms *top-down* and *bottom-up* have their roots in traditional structured design and structured programming. A traditional structured design decomposes software into a series of modules or functions, which are hierarchically related to one another. As a visual analogy, consider a typical organization chart with the president or CEO at the top. In structured design, a single module (the president or CEO) controls the entire software program. Modules at the bottom perform low-level specialized tasks when directed to do so by a module at the next higher level. **Top-down development** begins with the CEO and works downward. **Bottom-up development** begins with the detailed modules at the lowest level and works upward to the CEO.

Top-down and bottom-up program development can also be applied to object-oriented designs and programs, although a visual analogy isn't obvious with object-oriented diagrams. The key issue is method dependency—that is, which methods call which other methods. Within an object-oriented subsystem or class, method dependency can be examined in terms of navigation visibility, as discussed in Chapters 12 and 13.

The primary advantage of top-down development is that there is always a working version of a program. As each method or class is implemented, stubs for the methods or classes on the next lower level are added. At every stage of development, the program can be executed and tested, and its behavior becomes more complex and realistic as development proceeds.

■ Use-Case Driven

IPO, top-down, and bottom-up development are only starting points for creating implementation and iteration plans. Other factors that must be considered include use-case-driven development, user feedback, training, documentation, and testing. Use cases deserve special attention in determining development order because they are one of the primary bases for dividing a development project into iterations.

In most projects, developers choose a set of related use cases for a single iteration and complete analysis, design, implementation, and deployment activities. **Use-case-driven development** occurs when developers choose which use cases to focus on first based on such factors as minimizing project risk, efficiently using nontechnical staff, or deploying some parts of the system earlier than others. For example, use cases with uncertain requirements or high technical risks are typically addressed in early iterations. Addressing uncertain requirements requires usability and other testing by nontechnical development staff, and those staff members may only be available at certain times in the project.

User feedback, training, and documentation all depend heavily on the user interfaces of the system. Early implementation of user interfaces enables user training and the development of user documentation to begin early in the development process. It also gathers early feedback on the quality and usability of the interface. Note the important role that this issue played in the opening case of this chapter.

■ Source Code Control

Development teams need tools to help coordinate their programming tasks. A **source code control system (SCCS)** is an automated tool for tracking source code files and controlling changes to those files. An SCCS stores project source code files in a repository, and it acts the way a librarian would—that is, implements check-in and checkout procedures, tracks which programmer has which files, and ensures that only authorized users have access to the repository.

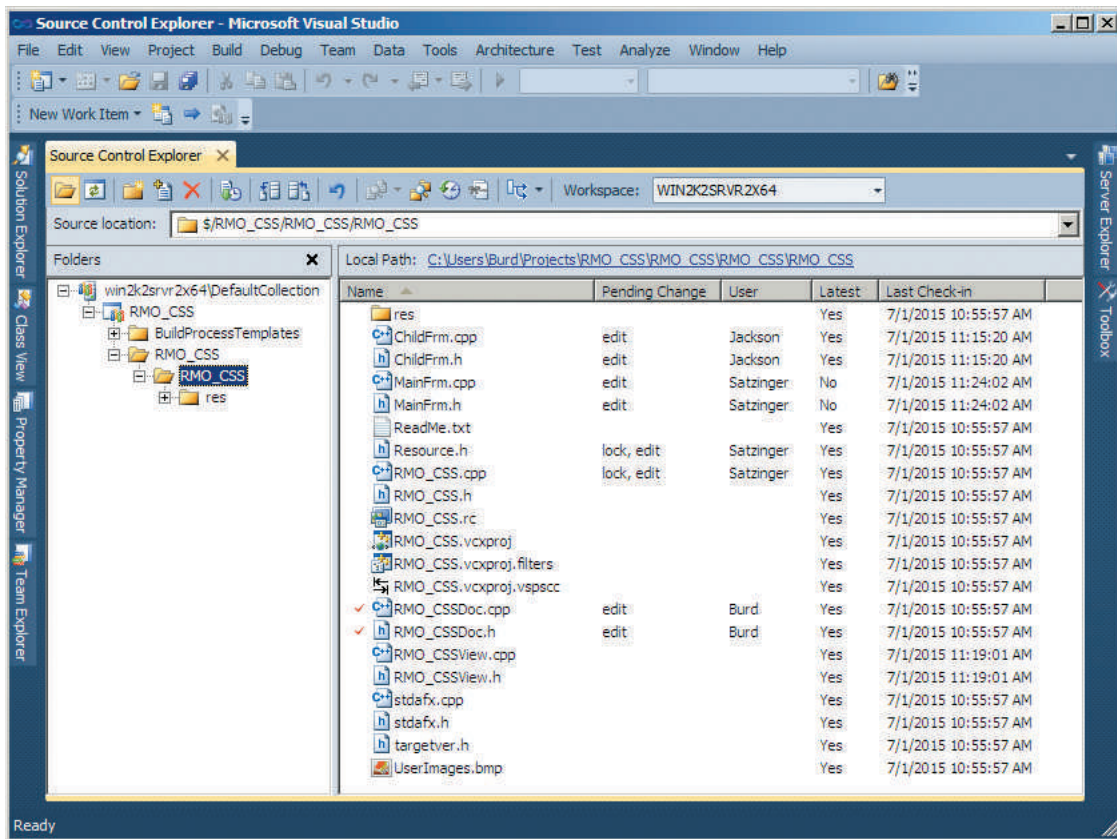
top-down development a development order that implements top-level modules first

bottom-up development a development order that implements low-level detailed modules first

use-case-driven development a development based on a selection of use cases to implement during project iterations

source code control system (SCCS) an automated tool for tracking source code files and controlling changes to those files

FIGURE 14-16 Project files managed by a source code control system



A programmer checks out a file in read-only mode when he or she wants to examine the code without making changes (e.g., to examine a module's interfaces to other modules). When a programmer needs to make changes to a file, he or she checks out the file in read/write mode. The SCCS allows only one programmer to check out a file in read/write mode. The file must be checked back in before another programmer can check it out in read/write mode.

Figure 14-16 shows the source code control display of Microsoft Visual Studio. Various source code files from the RMO CSS are shown in the display. Some files are currently checked out by programmers. For each file checked out in read/write mode, the program lists the programmer who checked it out, the date and time of checkout, and whether the copy currently stored in the central repository is the most current (latest) version.

An SCCS prevents multiple programmers from updating the same file at the same time, thus preventing inconsistent changes to the source code. Source code control is an absolute necessity when programs are developed by multiple programmers. It prevents inconsistent changes and automates coordination among programmers and teams. The repository also serves as a common facility for backup and recovery operations.

■ Packaging, Installing, and Deploying Components

As with the other disciplines discussed in this chapter, deployment activities are highly interdependent with activities of the other disciplines. In short, a system or subsystem can't be deployed until it has been implemented and tested. If a system or subsystem is large and complex, it is typically deployed in multiple stages or versions, thus necessitating some formal method of configuration and change management.

Important issues to consider when planning deployment include the following:

- Incurring costs of operating both systems in parallel
- Detecting and correcting errors in the new system
- Potentially disrupting the company and IS operations
- Training personnel and familiarizing customers with new procedures

Different approaches to deployment represent different trade-offs among cost, complexity, and risk. The most commonly used deployment approaches are:

- Direct deployment
- Parallel deployment
- Phased deployment

Each approach has different strengths and weaknesses, and no one approach is best for all systems. Each approach is discussed in detail in the following sections.

■ Direct Deployment

direct deployment or **immediate cutover** a deployment method that installs a new system, quickly makes it operational, and immediately turns off any overlapping systems

In a **direct deployment**, the new system is installed and quickly made operational, and any overlapping systems are then turned off. Direct deployment is also sometimes called **immediate cutover**. Both systems are concurrently operated for only a brief time (typically a few days or weeks) while the new system is being installed and tested. **Figure 14-17** shows a timeline for direct deployment.

The primary advantage of direct deployment is its simplicity. Because the old and new systems aren't operated in parallel, there are fewer logistical issues to manage and fewer resources required. The primary disadvantage of direct deployment is its risk. Because older systems aren't operated in parallel, there is no backup in the event that the new system fails. The magnitude of the risk depends on the nature of the system, the cost of workarounds in the event of a system failure, and the cost of system unavailability or less-than-optimal system function.

■ Parallel Deployment

parallel deployment a deployment method that operates the old and the new systems for an extended time period

In a **parallel deployment**, the old and new systems are operated for an extended period of time (typically weeks or months). **Figure 14-18** illustrates the timeline for parallel deployment. Ideally, the old system continues to operate until the new system has been thoroughly tested and determined to be error-free and ready to operate independently. As a practical matter, the time allocated for

FIGURE 14-17 *Direct deployment and cutover*

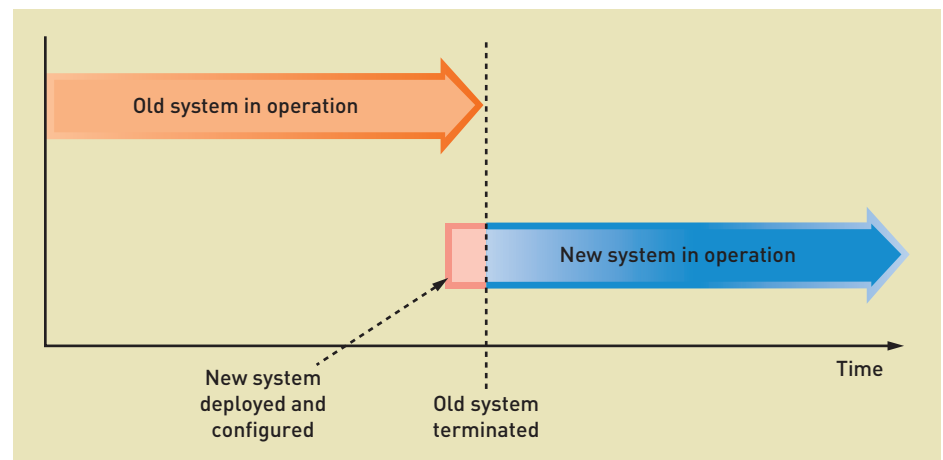
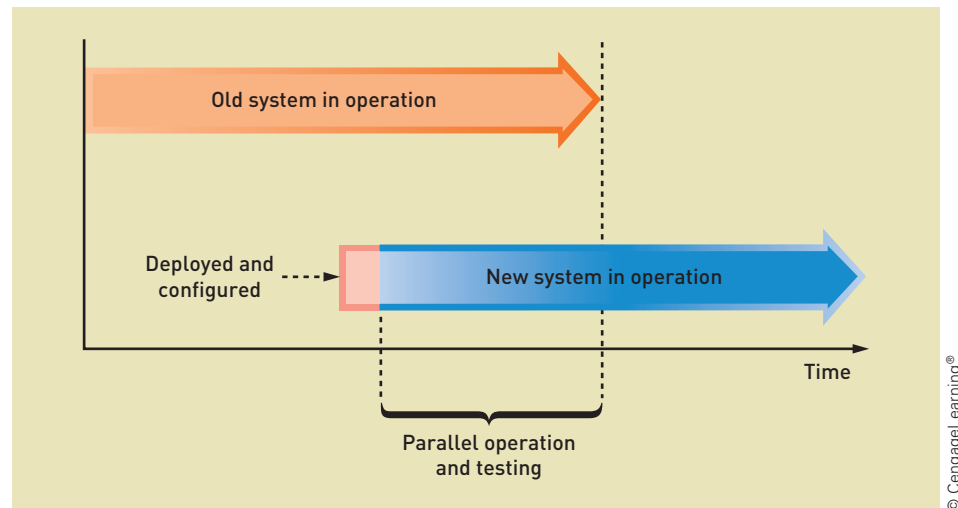


FIGURE 14-18 Parallel deployment and operation



parallel operation is often determined in advance and limited to minimize the cost of dual operation.

The primary advantage of parallel deployment is relatively low operational risk. If both systems are operated completely (i.e., using all data and exercising all functions), the old system functions as a backup for the new system. Any failure in the new system can be mitigated by relying on the old system.

The primary disadvantage of parallel deployment is cost. During the period of parallel operation, the organization pays to operate both systems. Extra costs associated with operating two systems in parallel include:

- Hiring temporary personnel or temporarily reassigning existing personnel
- Acquiring additional computing and network capacity
- Increasing managerial and logistical complexity

Parallel operation is generally best when the consequences of a system failure are severe. Parallel operation substantially reduces the risk of a system failure through redundant operation. The risk reduction is especially important for such mission-critical applications as customer service, production control, basic accounting functions, and most forms of online transaction processing.

Full parallel operation may be impractical for any number of reasons, including:

- Inputs to one system may be unusable by the other, and it may not be possible to use both types of inputs.
- The new system may use the same equipment as the old system (e.g., computers, I/O devices, and networks), and capacity may be insufficient to operate both systems.
- Staffing levels may be insufficient to operate or manage both systems at the same time.

When full parallel operation isn't possible or feasible, a partial parallel operation may be employed instead. Possible modes of partial parallel operation include:

- Processing only a subset of input data in one of the two systems. The subset could be determined by transaction type, geography, or sampling (e.g., every 10th transaction).
- Performing only a subset of processing functions (e.g., updating account history but not printing monthly bills).
- Performing a combination of data and processing function subsets.

Partial parallel operation always entails the risk that significant errors or problems will go undetected. For example, parallel operation with partial input increases the risk that errors associated with untested inputs won't be discovered.

■ Phased Deployment

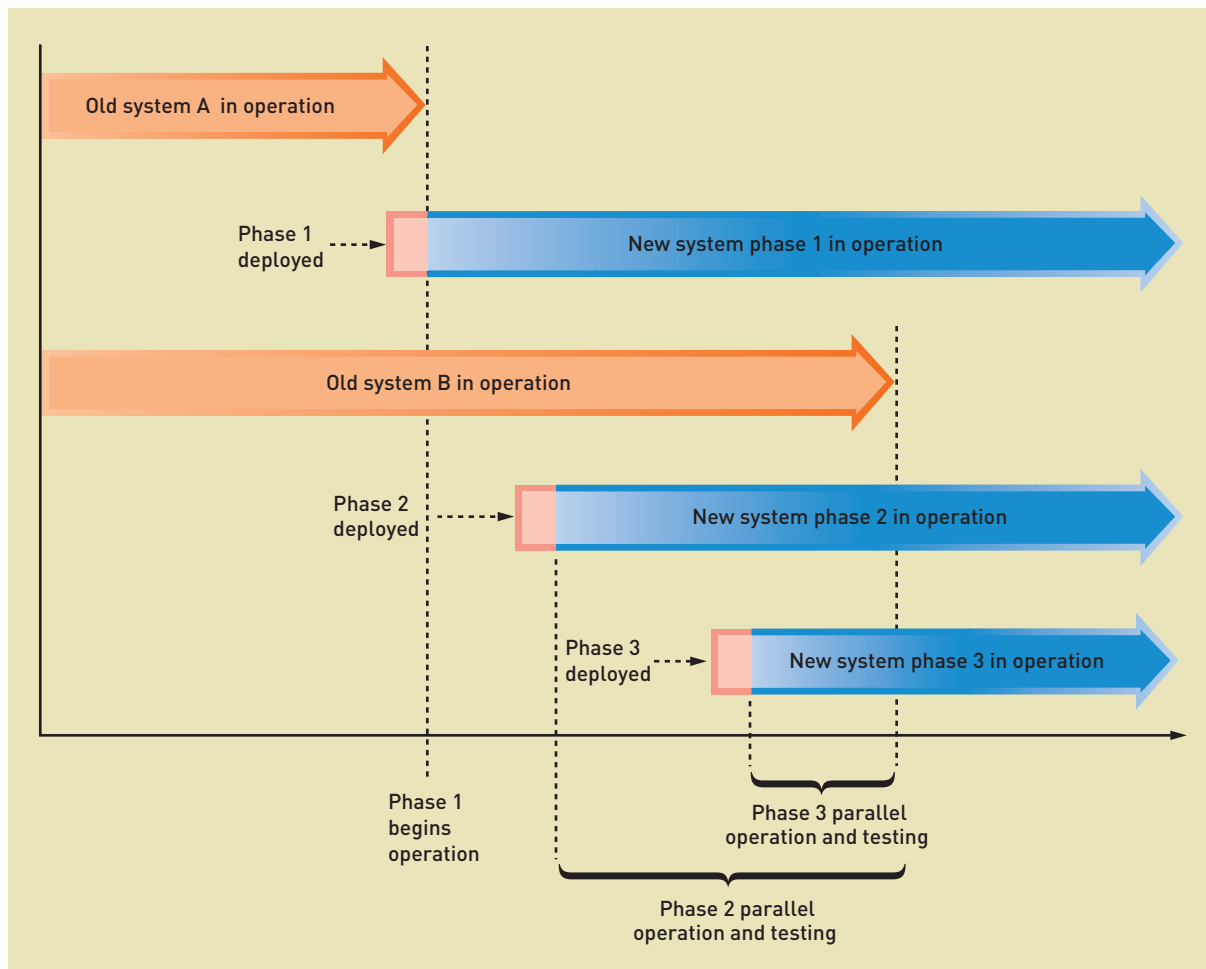
phased deployment a deployment method that installs a new system and makes it operational in a series of steps or phases

In a **phased deployment**, the system is deployed in a series of steps or phases. Each phase adds components or functions to the operational system. During each phase, the system is tested to ensure that it is ready for the next phase. Phased deployment can be combined with parallel deployment, particularly when the new system will take over the operation of multiple existing systems.

Figure 14-19 shows a phased deployment with direct and parallel deployment of individual phases. The new system replaces two existing systems. The deployment is divided into three phases. The first phase is a direct replacement of one of the existing systems. The second and third phases are different parts of a parallel deployment that replace the other existing system.

The primary advantage of phased deployment is reduced risk because failure of a single phase is less problematic than failure of an entire system. The primary disadvantage of phased deployment is increased complexity. Dividing the deployment into phases creates more activities and milestones, thus making the entire process more complex. However, each phase contains a smaller and more manageable set of activities. If the entire system is simply too big or complex to install at one time, the reduced risks of phased deployment outweigh the increased complexity inherent in managing and coordinating multiple phases.

FIGURE 14-19 Phased deployment with direct cutover and parallel operation



support activities the activities in the support phase whose objective is to maintain and enhance the system after it is installed and in use

■ Support Activities after Deployment

The predictive waterfall SDLC explicitly includes a support phase, but adaptive, iterative SDLCs typically don't. In fact, newer adaptive SDLCs consider support to be an entirely separate project worthy of its own support methodology.

The objective of the **support activities** is to keep the system running productively during the years following its initial deployment. They begin only after the new system has been installed and put into production, and they last throughout the productive life of the system. Most business systems are expected to last for years. During the support activities, upgrades or enhancements may be carried out to expand the system's capabilities, and these will require their own development projects. Three major activities occur during support:

- Maintaining the system
- Enhancing the system
- Supporting the users

Every system, especially a new one, contains components that don't function correctly. Software development is complex and difficult, so it is never free of error. Of course, the objective of a well-organized and carefully executed project is to deliver a system that is robust and complete and that gives correct results. However, because of the complexity of software and the impossibility of testing every possible combination of processing requirements, there will always be errors. In addition, business needs and user requirements change over time. Key tasks in maintaining the system include fixing the errors (also known as fixing bugs) and making minor adjustments to processing requirements. Usually, a system support team is assigned responsibility for maintaining the system.

Most newly hired programmer analysts begin their careers working on system maintenance projects. Tasks typically include changing the information provided in a report, adding an attribute to a table in a database, or changing the design of Windows or browser forms. These changes are requested and approved before the work is assigned, so a change request approval process is always part of the system support phase.

During the productive life of a system, it is also common to make major modifications. At times, government regulations require new data to be maintained or information to be provided. Also, changes in the business environment—new market opportunities, new competition, or new system infrastructure—necessitate major changes to the system. To implement these major modifications, the company must approve and initiate an upgrade development project. An upgrade project often results in a new version of the system. During your career, you may participate in several upgrade projects.

The next section describes the techniques used during the support activities to maintain the system. These techniques apply to the final testing and deployment activities as well as to the system after it is in production.

■ Change and Version Control

Though not formal activities of the implementation or deployment core processes, change and version control are key parts of managing software development, testing, deployment, and support activities. Medium- and large-scale systems are complex and constantly changing. Changes occur rapidly during implementation and more slowly during deployment and after the system is in use. System complexity and rapid change create a host of management problems, particularly for testing and postdeployment support.

Change and version control tools and processes handle the complexity associated with testing and supporting a system through multiple versions. Tools and processes are typically incorporated into implementation activities from the beginning and continue throughout the life of a system. Most organizations use a common set of tools and procedures for all their systems.

■ Versioning

Complex systems are developed, installed, and maintained in a series of versions to simplify testing, deployment, and support. It isn't unusual to have multiple versions of a system deployed to end users and yet more versions in different stages of development. A system version created during development is called a *test version*. A test version contains a well-defined set of features and represents a concrete step toward final completion of the system. Test versions provide a static system snapshot and a checkpoint to evaluate the project's progress.

alpha version a test version that is incomplete but ready for some level of rigorous integration or usability testing

beta version a test version that is stable enough to be tested by end users over an extended period of time

production version, release version, or production release a system version that is formally distributed to users or made operational for long-term use

maintenance release a system update that provides bug fixes and small changes to existing features

An **alpha version** is a test version that is incomplete but ready for some level of rigorous integration or usability testing. Multiple alpha versions may be built depending on the size and complexity of the system. The lifetime of an alpha version is typically short—days or weeks.

A **beta version** is a test version that is stable enough to be tested by end users over an extended period of time. A beta version is produced after one or more alpha versions have been tested and known problems have been corrected. End users test beta versions by using them to do real work. Thus, beta versions must be more complete and less prone to disastrous failures than alpha versions. Beta versions are typically tested over a period of weeks or months.

A system version created for long-term release to users is called a **production version, release version, or production release**. A production version is considered a final product, although software systems are rarely “finished” in the usual sense of that term. Minor production releases (sometimes called **maintenance releases**) provide bug fixes and small changes to existing features. Major production releases add significant new functionality and may be the result of rewriting an older release from the ground up.

Keeping track of versions is complex. The development teams must coordinate existing production versions with its upgrades with new versions. **Figure 14–20** illustrates some of the overlap that occurs and must be coordinated. Each version needs to be uniquely identified for developers, testers, and users. In applications designed to run under Windows, users typically view the version information by choosing the About item from the standard Help menu (see **Figure 14-21**). Users seeking support or reporting errors in a beta or production version use this feature to report the system version to testers or support personnel.

Controlling multiple versions of the same system requires sophisticated version control software, which is often built in to development tools or can be obtained through a separate source code and version control system, as described later in this chapter. Programmers and support personnel can extract the current version or any previous version for execution, testing, or modification. Modifications are saved under a new version number to protect the accuracy of the historical snapshot.

Beta and production versions must be stored as long as they are installed on any servers or user machines. Stored versions are used to evaluate future bug reports. For example, when a user reports a bug in version 1.0, support personnel extract that release from the archive and attempt to replicate the user's error. Feedback provided to the user is specific to version 1.0, even if the most recent production release is a higher-numbered version.

■ Submitting Error Reports and Change Requests

To manage the risks associated with change, most organizations adopt formal control procedures for all systems under development and in operation. Formal controls are designed to ensure that potential changes are adequately described, considered, and planned before being implemented and deployed. Typical change control procedures include these:

- Standard reporting methods
- Review of requests by a project manager or change control committee
- For operational systems, extensive planning for design and implementation

FIGURE 14-20 Need for version control

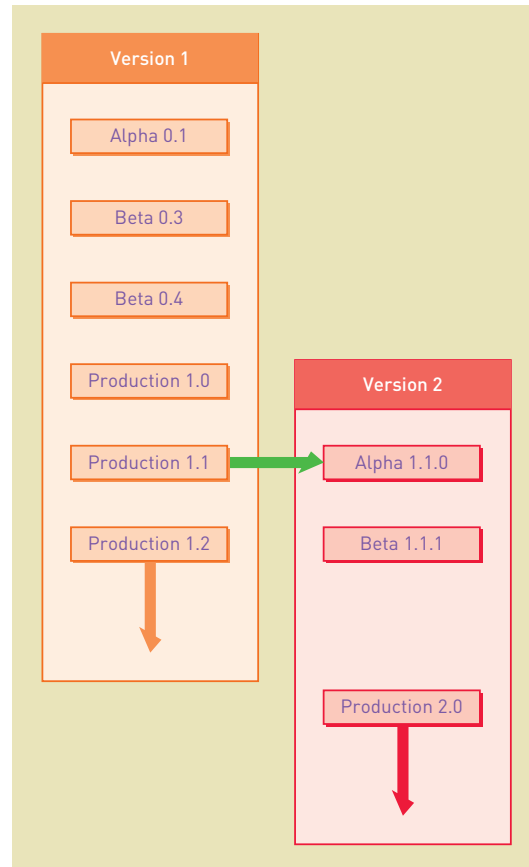


FIGURE 14-21 About box of a typical Windows application

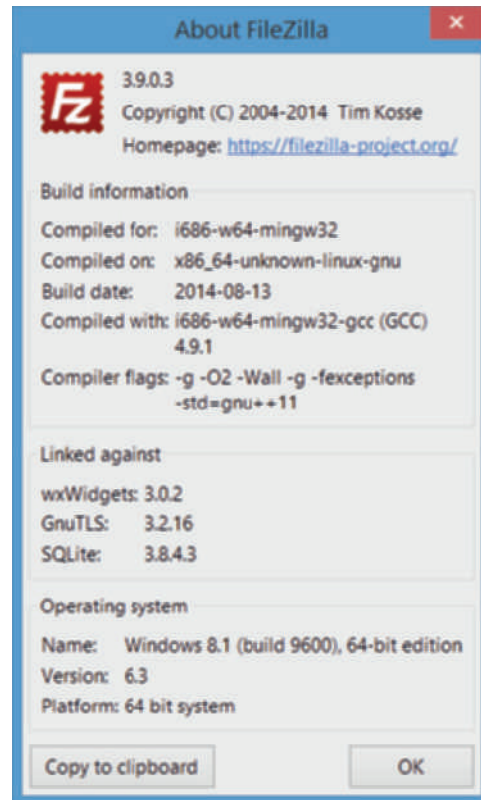
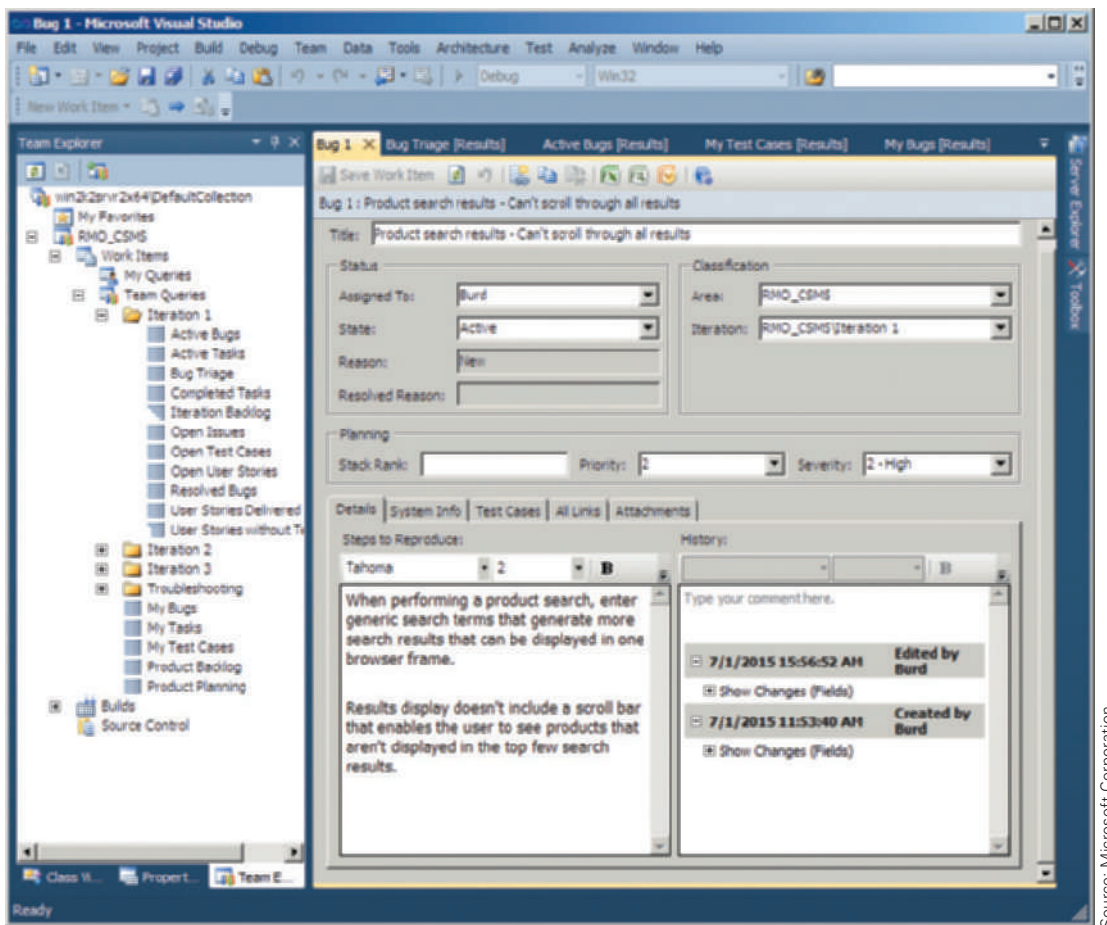


FIGURE 14-22 Sample error report in Microsoft Visual Studio



Source: Microsoft Corporation

Figure 14-22 shows a sample error (bug) report that has been completed by a tester or system developer. In this case, error reporting is integrated into the application development tool and source code control system, which enables the project manager to centrally manage all reports, assign reports to specific developers, and track each report to its resolution.

Similar tools can be used to report and manage errors and requests for new features in operational systems. In the case of new features, the request is usually submitted to a change control committee that reviews the change request to assess the impact on existing computer hardware and software, system performance and availability, security, and operating budget. Approved changes are added to the list of pending changes for budgeting, scheduling, planning, and implementation.

■ Implementing a Change

Change implementation follows a miniature version of the SDLC. Most of the SDLC activities are performed, although they may be reduced in scope or sometimes completely eliminated. In essence, a change for a maintenance release is an incremental development project in which the user and technical requirements are fully known in advance. Analysis activities are typically skimmed or skipped, design activities are substantially reduced in scope, and the entire project is typically completed in one or two short iterations.

Planning for a change includes these activities:

- Identify what parts of the system must be changed.
- Secure resources (such as personnel) to implement the change.
- Schedule design and implementation activities.
- Develop test criteria and a testing plan for the changed system.

production system the version of the system used daily to support organizational operations

test system a copy of the production system that is modified to test changes

Whenever possible, changes are implemented and tested on a copy of the operational system. The **production system** is the version of the system used day to day. The **test system** is a copy of the production system that is modified to test changes. The test system may be developed and tested on separate hardware or on a redundant system. The test system becomes the operational system only after complete and successful testing.

■ Putting It All Together—RMO Revisited

In a medium-sized or large-scale development project, managers usually feel overwhelmed by the sheer number of activities to be performed, their interdependencies, and the risks involved. This section gives you a glimpse of the interplay among those issues by showing how Barbara Halifax’s team developed an iteration plan for RMO’s Customer Support System (CSS). But keep in mind that no single example can adequately prepare you to tackle iteration planning for a complex project. That is why iteration planning and other project planning tasks are typically performed by developers with years of experience.

Before reading the rest of this section, you may want to review earlier descriptions of the RMO case in Chapters 2, 3, 4, 6, and 11. Some basic parameters for the project are already described, including subsystem boundaries and a 48-week project length (nine 4-week iterations).

Chapter 11 describes Barbara’s early planning decisions. In this section, she expands on those decisions, makes some changes to her earlier decisions, makes additional key decisions, and develops the revised iteration plan shown in **Figure 14-23**. The sections that follow describe key issues and decisions that underlie that plan.

■ Upgrade or Replace?

Upgrading the current CSS “in place” was ruled out early in project planning for these reasons:

- The current infrastructure is near capacity.
- RMO expects to save money by having an external vendor host the CSMS.
- Existing CSS programs and Web interfaces are a hodgepodge developed over 15 years.
- Current system software is several versions out of date.
- Infrastructure that supports the current CSS can be repurposed to expand SCM capacity.

In short, it would be too complex to upgrade the current CSS without disrupting operations, and the risks of upgrading old infrastructure and application software are simply too great. By building and deploying an entirely new system, RMO will make a clean break from the existing CSS and its supporting infrastructure. A new hosted infrastructure will be developed for the CSMS. After the first deployment phase, the existing CSS infrastructure will be updated to match the hosted environment and serve as a test environment for later development and deployment activities.

FIGURE 14-23 Revised CSMS iteration plan for RMO

Iteration	Description
1	Define business models and development/deployment environment. Define essential use cases and rough class diagram. Storyboard sales processing. Finalize deployment environment. Select and acquire network components, system software, hardware, and development tools. Create a CSS database copy with minimal data content as a starting point for CSMS database. Construct a simple prototype for adding a customer order (no database updates) and perform usability testing.
2	Define class, use case, sequence diagrams, and programs, concentrating on the key use cases (<i>Search for item</i> , <i>Fill shopping cart</i> , <i>Check out shopping cart</i> , <i>Look up customer</i> , and <i>Create customer account</i>). Deploy infrastructure components, including operating systems, Web/application servers, and DBMS by the middle of the iteration. Update database schema based on newly defined or revised classes and associations. Perform usability, unit, and integration testing to validate database design, customer/sales function set, and user interfaces.
3	Loop through iteration 2 use cases again and make all changes determined at the end of the previous iteration. Expand requirements and design to cover additional sales use cases and essential customer account and order-fulfillment use cases. Perform usability, unit, and integration testing.
4	Loop through iteration 3 use cases again and make all changes determined at the end of the previous iteration. Expand requirements and design to cover remaining Marketing subsystem use cases for products and promotions. Develop customer-oriented online help for all functions implemented in previous iterations. Prepare training materials and conduct training for phone and retail stores sales personnel. Finalize the new database and prepare it for data migration. Develop data migration (import) procedures. Test and refine data migration procedures by importing all data from the CSS database.
5	Loop through iteration 4 use cases again and make all changes determined at the end of the previous iteration. Continue training for phone and retail stores sales personnel. Conduct usability tests with a large number of actual or simulated customers. Make any needed changes to user interfaces, including online help. Conduct performance and stress testing and make any needed changes. Create a copy of the CSMS deployment environment at the Park City data center for use as a test system for version 2.0 development. Conduct user acceptance testing. Import all CSS database changes since the last import. Place version 1.0 into production.
6	Monitor system performance and user comments. Develop a change list and classify them as "ASAP" or "version 2.0." Implement ASAP changes. Expand requirements and design to cover essential use cases from the Reporting subsystem and those related to social networking. Migrate database updates from CSMS to CSS database twice per day. If no problems are encountered with CSMS, discontinue data migration and old system operation at the end of this iteration.
7	Loop through iteration 6 use cases again and make all changes determined at the end of the previous iteration. Expand requirements and design to cover all remaining use cases. Update database design as needed to support version 2.0 use cases. Program iteration 7 and use cases and conduct unit and integration testing.
8	Develop customer-oriented online help for all functions implemented in iterations 6 and 7. Prepare training materials and conduct training for sales, marketing, and management personnel. Conduct usability tests with a large number of actual or simulated customers. Make any needed changes to user interfaces, including online help. Update the production database with any structural changes in the test database.
9	Continue training for sales, marketing, and management personnel. Conduct performance and stress testing and make any needed changes. Conduct user acceptance testing. Place version 2.0 into production.

© CengageLearning®

■ Phased Deployment to Minimize Risk

The schedule described in Chapter 11 didn't call for phased deployment, but neither did it directly consider such deployment issues as database development, data migration, and training. To minimize deployment risks, the CSMS will be deployed in two versions. Version 1.0 will reimplement most of the existing CSS use cases with minimal changes. Version 2.0 will incorporate bug fixes and incremental improvements to version 1.0 and will add additional functionality not present in the CSS, including social networking, feedback/recommendations, business partners, and Mountain Bucks.

The two-phase deployment minimizes project risk by dividing a single large deployment into two smaller deployments. Another key risk mitigation feature is maintaining the current CSS and its database as a backup for at least one iteration after version 1.0 deployment. If a serious problem arises with version 1.0, RMO can revert to the current CSS simply by redirecting Web site accesses back to its internal servers.

■ Database Development and Data Conversion

Many of the classes in the CSMS class diagram are already represented in the existing CSS database. However, there are some new classes and associations and some changes to existing classes. Thus, there is some degree of compatibility between the old and new databases, but not enough to enable an upgraded version of the current database to directly interface with both systems. Thus, a new CSMS database will need to be built, and data will need to be migrated from the CSS database prior to deploying version 1.0.

Database development and migration prior to version 1.0 deployment will occur over multiple iterations. The iteration plan calls for creating a copy of the CSS database early in the project and making incremental changes to it. All data in the production CSS database will be migrated to the CSMS database near the end of the fourth iteration. If problems are encountered, they will be resolved and the migration will be repeated as early as possible during the fifth iteration. Migrating much of the data during the fourth iteration will enable fifth-iteration testing of user interfaces with real data from real customers and products and system and stress testing with a “production sized” database.

At the end of the fifth iteration, all CSS database changes since the last full migration will be copied to the CSMS database. Copying only the changes will enable migration within a matter of hours. The CSS system will be offline during the migration. Cutover to the CSMS will occur as soon as the migration is completed. To minimize risk, additional data conversion routines will copy new data from the CSMS database back to the CSS database twice per day during the fifth iteration. If disaster strikes, the CSS can be restarted with a current and complete database. If CSMS version 1.0 passes all user acceptance tests during the fifth iteration, the CSS will be turned off and data migration will cease.

■ Development Order

A combination of IPO and use-case-driven development order is the primary basis for the development plan. By starting with a copy of the CSS database, a set of test data will exist from the first iteration, thus enabling the highest-risk use cases to be tackled first. These involve the entire Sales subsystem and customer-facing portions of the Order Fulfillment subsystem. The risks arise from new technology, uncertainty about requirements, and the operational importance of sales and order fulfillment to RMO. By tackling those use cases first, Barbara allowed her development staff plenty of time to resolve uncertainties and test related software. Note that significant testing of these functions began in iteration 2 and continued through most of the project.

■ Documentation and Training

Training activities were spread throughout later project iterations for both production versions. Initial training exercises covered the highest-risk portion of the system prior to deployment. They also enabled developers to do integration and performance testing on the sales-related use cases long before deployment. Additional training continued as new functions were added to the system, providing a gradual ramping up of user skills and developer workload.

CHAPTER SUMMARY

Implementation and deployment are complex processes because they consist of so many interdependent activities. Testing is a key activity of implementation and deployment. Software components must be constructed in an order that minimizes the use of development resources and maximizes the ability to test the system and correct errors. Unfortunately, those two goals often conflict. Thus, a program development plan is a trade-off among available resources, available time, and the desire to detect and correct errors prior to system deployment.

Configuration and change management activities track changes to models and software through multiple system versions, which enables developers to test and deploy a system in stages. Versioning also improves postdeployment support by enabling developers to track problem support to specific system versions. Source code control systems enable development teams to coordinate their work.

KEY TERMS

alpha version	parallel deployment	system documentation
beta version	performance test or stress test	system test
bottom-up development	phased deployment	test case
build and smoke test	production system	test data
direct deployment or immediate cutover	production version, release version, or production release	test system
driver	response time	throughput
input, process, output (IPO) development order	source code control system (SCCS)	top-down development
integration test	stub	unit test
maintenance release	support activities	use-case-driven development
		user acceptance test (UAT)
		user documentation

REVIEW QUESTIONS

- List and briefly describe each activity of the SDLC core processes Build, test, and integrate system components and Complete system tests and deploy solution.
- What is the purpose of unit testing? Who performs it? How is the test data prepared?
- What is the difference between unit testing and integration testing?
- What are the objectives of integration testing?
- What distinguishes integration testing from system testing? What kinds of tests can be included in system testing?
- Who is responsible for user acceptance testing (UAT)? What are the primary objectives of UAT?
- What is a good way to identify test cases to be used for UAT?
- Who defines the test data for the UAT?
- What is a test case? What are the characteristics of a good test case?
- What are the various elements required for good test data?
- What is a driver? What is a stub? With what type of test is each most closely associated?
- During what types of testing would it be helpful to have an error tracking log?
- List possible sources of data used to initialize a new system database. Briefly describe the tools and methods used to load initial data into the database.
- How do user documentation and training activities differ between end users and system operators?
- List the major items in the architecture environment that must be configured for production systems.
- List and briefly describe four basic approaches to program development order. What are the advantages and disadvantages of each?

17. How can the concepts of top-down and bottom-up development order be applied to object-oriented software?
18. Why has use-case-driven development become popular with iterative development?
19. What is a source code control system? Why is such a system necessary when multiple programmers build a program or system?
20. Briefly describe direct, parallel, and phased deployments. What are the advantages and disadvantages of each deployment approach?
21. Define the terms alpha version, beta version, and production version. Are there well-defined criteria for deciding when an alpha version becomes a beta version or a beta version becomes a production version?

PROBLEMS AND EXERCISES

1. Describe the process of testing software developed with the IPO (input, process, output), top-down, bottom-up, and use-case-driven development orders. Which development order results in the fewest resources required for testing? What types of errors are likely to be discovered earliest under each development order? Which development order is best, as measured by the combination of required testing resources and ability to capture important errors early in the testing process?
2. Assume that you and three of your classmates are charged with developing the first prototype to implement the RMO use case *Create/update customer account*. Create a development and testing plan to write and test the classes and methods. Assume that you have two weeks to complete all tasks.
3. Talk with a computer center or IS manager about the testing process used with a recently deployed system or subsystem. What types of tests were performed? How were test cases and test data generated? What types of teams developed and implemented the tests?
4. Consider the issue of documenting a system by using only electronic models developed with an integrated development tool, such as Microsoft Visual Studio or Oracle JDeveloper. The advantages are obvious (e.g., the analyst modifies the models to reflect new requirements and automatically generates an updated system). Are there any disadvantages? (Hint: The system might be maintained for a decade or more.)
5. Talk with an end user at your school or work about the documentation and training provided with a recently installed or distributed business application. What types of training and documentation were provided? Did the user consider the training to be sufficient? Does the user consider the documentation to be useful and complete?
6. Assume you are in charge of implementation and deployment of a new system that is replacing a critical existing system that is used 24 hours a day. To minimize risk, you plan to phase in deployment of new subsystems over a period of six weeks and operate both systems in parallel for at least three weeks beyond the last new subsystem deployment. Because there aren't enough personnel to operate both systems, you plan to hire up to 30 temporary workers during the parallel operation period. How should you use the temporary workers? In answering that question, be sure to consider these issues:
 - a. Some current personnel will be trained before subsystem deployments, and those employees will train other employees.
 - b. Employees newly trained on the system will probably not reach their former levels of efficiency for many weeks.

CASE STUDY

Hudsonbanc Billing System Upgrade

Two regional banks with similar geographic territories merged to form HudsonBanc. Both banks had credit card operations and operated billing systems that had been internally developed and upgraded over three decades. The systems performed similar functions, and both operated

primarily in batch mode on mainframe computers. Merging the two billing systems was identified as a high-priority cost-saving measure.

HudsonBanc initiated a project to investigate how to merge the two billing systems. Upgrading either system was quickly ruled out because the existing technology was considered old and the costs of upgrading the system

were estimated to be too high. HudsonBanc decided that a new component-based, Web-oriented system should be built or purchased. Management preferred the purchase option because it was assumed that a purchased system could be brought online more quickly and cheaply. An RFP (request for proposal) was prepared, many responses were received, and after months of business modeling and requirements activities, a vendor was chosen.

Hardware for the new system was installed in early January. Software was installed the following week, and a random sample of 10 percent of the customer accounts was copied to the new system. The new system was operated in parallel with the old systems for two months. To save costs involved with complete duplication, the new system computed but didn't actually print billing statements. Payments were entered into both systems and used to update parallel customer account databases. Duplicate account records were checked manually to ensure that they were the same.

After the second test billing cycle, the new system was declared ready for operation. All customer accounts were migrated to the new system in mid-April. The old systems were turned off on May 1, and the new system took over operation. Problems occurred almost immediately. The system was unable to handle the greatly increased volume of transactions. Data entry and customer Web access slowed to a crawl, and payments were soon backed up by several weeks. The system wasn't handling certain types of transactions correctly (e.g., charge corrections and credits for overpayment). Manual inspection of the recently

migrated account records showed errors in approximately 50,000 accounts.

It took almost six weeks to adjust the incorrect accounts and update functions to handle all transaction types correctly. On June 20, the company attempted to print billing statements for the 50,000 corrected customer accounts. The system refused to print any information for transactions more than 30 days old. A panicked consultation with the vendor concluded that fixing the 30-day restriction would require more than a month of work and testing. It was also concluded that manual entry of account adjustments followed by billing within 30 days was the fastest and least risky way to solve the immediate problem.

Clearing the backlog took two months. During that time, many incorrect bills were mailed. Customer support telephone lines were continually overloaded. Twenty-five people were reassigned from other operational areas, and additional phone lines were added to provide sufficient customer support capacity. System development personnel were reassigned to IS operations for up to three months to assist in clearing the billing backlog. Federal and state regulatory authorities stepped in to investigate the problems. HudsonBanc agreed to allow customers to spread payments for late bills over three months without interest charges. Setting up the payment arrangements further aggravated the backlog and staffing problems.

1. What type of installation did HudsonBanc use for its new system? Was it an appropriate choice?
2. How could the operational problems have been avoided?

RUNNING CASE STUDIES

Community Board of Realtors®

Assume that the Multiple Listing Service that is under development will replace an existing system developed many years ago. The database requirements and design for the old and new systems are very similar. Unfortunately, the existing system stores its data in a Microsoft Access database, which provides little support for simultaneous access and updates by multiple users. An important reason for replacing the current system is to upgrade to a DBMS that can easily support many simultaneous accesses.

The current plan is to use Microsoft SQL Server as the new DBMS and to migrate all data from the

existing Microsoft Access database immediately prior to full deployment. Perform these tasks to prepare for this migration:

1. Investigate data migration from Microsoft Access to SQL Server. What tools are available to assist in or perform the migration? If there are multiple possible tools, which should you use and why?
2. Develop plans to test the migration tools/strategy in advance of full deployment. When should the test be performed, and how will you determine whether the test has been "passed"?

The Spring Breaks 'R' Us Travel Service

Review the case-related questions and tasks as well as your responses from Chapters 10 and 11. As described in previous chapters, assume the new system will

upgrade an existing system and add new social networking functions to it. Specifically, review your answer to question 2 in Chapter 11 in light of the more

detailed understanding of the risks, costs, and benefits of various implementation orders and deployment approaches that you gained by reading this chapter.

1. For each subsystem—Resort Relations, Student Booking, Accounting and Finance, and Social Networking—specify which other subsystem(s) it depends on for input data.

On the Spot Courier Services

In Chapter 10, we identified these four subsystems:

- Customer Account subsystem (such as Customer Account)
- Pickup Request subsystem (such as Sales)
- Package Delivery subsystem (such as Order Fulfillment)
- Routing and Scheduling subsystem

In Chapter 10, you also decided on a development order for these four subsystems, assuming a single two-person team. In Chapter 11, you created individual subsystem iteration schedules and a combined project schedule. In Chapter 7, you identified equipment that would be needed for the system.

Your assignment for this chapter is to develop a test plan for each subsystem and for the overall project as well as to develop a conversion/deployment schedule.

1. For your test plan, do the following:
 - a. Develop an iteration test plan (i.e., one that applies to and can be used within a subsystem iteration mini-project). Discuss which types of testing (as identified in this chapter) you would include and why. Estimate how much time will be needed for each type of test. Discuss what types of testing might be combined or scheduled with an overlap.

Sandia Medical Devices

Refer to the case information provided at the end of Chapters 10 and 11 and the domain class diagram at the end of Chapter 9. Review and update your results from performing the tasks at the end of Chapter 11 based on the information provided in this chapter. Then, answer these questions:

1. What integration and system tests are required, and when should they be incorporated into the iteration schedule?

2. Can the four subsystems be developed and deployed independently? If so, in which order should they be developed and deployed? If not, explain why not and describe how you would develop and deploy the system.

- b. Develop a total project test plan to integrate all the subsystems. Discuss which types of testing you would include and why. (Don't put them on a schedule yet.)
2. Develop a conversion/deployment plan. Discuss these:
 - a. Data conversion: Which parts of the data must be saved from the old spreadsheet/manual system? Which parts of the data can just be discarded (i.e., not moved to the new system)? Discuss specific tables that you identified in Chapter 9.
 - b. Deployment: Based on your decisions about which subsystems should be deployed first (Chapter 10), your overall testing plan, and your data conversion decisions, develop an overall schedule for testing and deployment of the new system. How would you characterize your solution: direct, parallel, or phased conversion? Support your answer by discussing the logic behind your decisions.
 3. Develop and discuss your current recommendation for hosting the system. Add to your deployment schedule the activities to set up the hosting environment. Include the steps to purchase equipment if needed.

2. What are the documentation and user training requirements for the system, and when should they be incorporated into the iteration schedule?
3. Assume that after deployment and a three-month testing and evaluation period, updates to the first Android-based system (client and server) will be implemented and another client-side version will be implemented for the iPhone. Develop an iteration plan for implementing and deploying the second version of the system.

FURTHER RESOURCES

Robert V. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley, 2000.

Mark Fewster and Dorothy Graham, *Software Test Automation*. Addison-Wesley, 1999.

Jerry Gao, H.-S. Jacob Tsao, and Ye Wu, *Testing and Quality Assurance for Component-Based Software*. Artech House Publishers, 2003.

William Horton, *Designing and Writing Online Documentation: Hypermedia for Self-Supporting Products* (2nd ed.). John Wiley & Sons, 1994.

William Horton, *Designing Web-Based Training: How to Teach Anyone Anything Anywhere Anytime*. John Wiley & Sons, 2000.

William Horton, *e-Learning by Design*. Pfeiffer, 2011.

International Association of Information Technology Trainers (ITrain) Web site, <http://itrain.org>.

David Yardley, *Successful IT Project Delivery*. Addison-Wesley, 2002.

INDEX

- 1NF(first normal form), 273–274
- 2NF (second normal form), 275–276
- 3NF (third normal form), 276–278
- abstract classes, 109–110, 379
- acceptance criteria, 72
- acceptance tests, 26, 316, 451
- access controls, 174–175
- accounting and financial management (AFM) systems, defined, OL-8
- action-expressions, 117, 119
- activation lifelines, 409
- activity diagrams
 - analysis and design models, 162
 - defined, 60
 - documenting workflows with, 60–63
 - for *Fill Shopping Cart*, 411
 - relationship with other models, 148
 - for RMO Supplier Information Subsystem, 24–25
 - for RMO Tradeshow system, 12
 - for use cases, 137–139
- activity-data matrix, defined, OL-48
- activity-location matrix, defined, OL-48
- actors, defined, 76, 81, 135
- adapter pattern, 428–430
- adaptive approaches to the SDLC, 297–298, 300. *See also* Agile development; project management
- affordance, 223
- aggregation, 110
- Agile development
 - defined, 8, 304
 - iterative, 8–9, 27
 - modeling principles, 305–307
 - philosophy and values, 304–305
 - Scrum, 316–318
 - UP, 308–312
 - XP, 312–316
- Agile modelling (AM), 305–307
- Agile project management (APM), 332–335. *See also* project management
- Ajax Corporation, 296
- alpha versions, 468
- alt frames, 142–143
- AM (agile modeling), 305–307
- analysis. *See* systems analysis
- analysis-related careers, OL-13–OL-15
- API (application programming interface). *See* application programming interface (API)
- APM (Agile project management), 332–335. *See also* project management
- application architecture and software, 191, 196, 451, 457
 - concepts, 195–200
 - defined, 40
 - design key question, 163
- application components
 - boundaries, 208
 - defined, 165
 - integration, 210–213
 - RMO CSMS application architecture, 208–210
- application programming interface (API), 201, 206, 259
- approval, quantifying factors for, 337–343
- apps, defined, 4
- architectural concepts, 195–200
 - distributed, 198–200
 - client/server architecture, 198–199
 - three-layer architecture, 199–200
 - software as a service, 196
 - Web service, 197–198
- architectural diagrams, 201–203
 - deployment, 202–203
 - location, 201–202
 - network, 202
- architectures
 - distributed databases, 279–284
 - hardware and network, 261
 - overview, 186–187
 - RMO CSMS application, 208–210
 - RMO's current technology architecture, 179–180
 - of RMO Supplier Information Subsystem, 24–25
 - of RMO Tradeshow System, 12, 16, 23
 - updated RMO technology, 207
- artifacts, defined, 401
- association classes, 105
- associations. *See also* relationships
 - among things in problem domain, 98–100
 - representing in database, 268–270
- assumptions sequence diagrams, 417
- asymmetric key encryption, 177
- attributes
 - class-level, 379
 - defined, 97, 263
 - in databases, 263
 - elaboration of, 380
 - of things in problem domain, 97–98
 - value, 263
- authentication, 174
- authorization, 174
- automation boundary, 81
- automation boundary, defined, OL-7
- availability, item, 79
- Aviation Electronics, 218
- backlog, product, 317
- backup, database, 172
- balancing, defined, OL-36
- bar graph sample, 250
- benefits, anticipated, 337, 340–343
- beta versions, 468
- binary associations, defined, 100
- black hole, defined, OL-36
- Blue Sky Mutual Funds, 326
- bottom-up development, 462
- boundary classes, 376–377
- brainstorming technique, 95–96
- break-even point, 342
- brief use case descriptions, 81, 133
- browser-based systems. *See also* Internet, Web-based applications
 - affordance and visibility in, 223–224
 - internal network, 391, 418
 - pros and cons of, 23
 - user interface guidelines, 240–242
- build and smoke testing, 451
- business benefits, 337
- business intelligence system, defined, OL-9
- business knowledge, for systems analyst, OL-12
- business logic layer. *See* domain layer
- business processes
 - elementary, 74
 - observing and documenting, 56–57
 - questions related to, 50–51
- camelback or camelcase notation, 103
- candidate key, 264
- cardinality, defined, 99
- ceremony, project management and, 330–331
- certificates, 177–179
- certifying authorities, 178
- change and version control, 467–471
- chaordic, defined, 304
- check boxes, 224
- class diagrams. *See also* design class diagrams (DCDs); domain model class diagrams
 - defined, 17, 103
 - for RMO Tradeshow System, 18
- class responsibility collaboration (CRC) cards
 - defined, 382
 - designing, 382–388
- classes. *See also* design class diagrams (DCDs); use case realization
 - abstract and concrete, 109, 379
 - on CRC cards, 382–388
 - creating tables for, 266
 - defined, 103
 - design overview, 376
 - notation for, 377–379
 - types of, in UML, 376
- classification hierarchies, representing, 270–271
- class-level attributes, 379
- class-level methods, 378
- clients (customers). *See also* stakeholders
 - in Agile projects, 304–305
 - defined, 329
 - reviewing with and obtaining approval from, 344–345
 - in Scrum, 317
 - in XP, 314
- clients (software/hardware), 48
- client/server architecture, 198–199. *See also* three-layer client-server architecture
- closed-ended questions, 51
- code ownership in XP, 314
- coding standards adoption in XP, 315
- cohesion, 392
- collaboration on CRC cards, 382–388
- collaboration support system (CSS), defined, OL-9
- colors, 97, 159, 229
- combo boxes, 239
- committees, oversight, 329–330
- communication diagrams, 399, 401–408
 - object-oriented design with, 403–408
 - extend input messages, 404–407
 - final design class diagram, 407–408
 - input models, 404
 - understanding, 401–403
 - use case realization with, 401–408

- communication, network, OL-47–OL-49
- communications
 - establishing environment for, 345–350
 - on Internet, 418
 - management of, 331
 - in XP, 312
- completeness controls, 171
- complex data types, 278
- complex update controls, 285–286
- component diagrams, 162
- composition, defined, 110
- compound attributes, defined, 98
- computer applications, defined, 4
- computing devices, 188–189
- concrete classes, 109, 379
- concurrency (concurrent states), 117–118
- concurrency in state machine diagrams, 117–118
- concurrent paths
 - activity diagram, 62
 - defined, 117
- consistency, 224–226
- Consolidated Concepts, 296
- construction phase, 309
- context diagram, defined, OL-26
- continuous integration, 314
- contracts for Agile projects, 305
- control break, 247
- controller classes, 377
- controller pattern, 428–429
- controls, integrity, 170–173
- controls, security. *See* security and system controls
- corrective actions, taking, 355–356
- cost/benefit analysis, 341–342
- costs
 - estimated, 340
 - intangible, 343
 - management of, 331, 334–335
 - coupling, 391–392
- CPM charts, OL-58–OL-62, OL-69–OL-70
- CRC (class responsibility collaboration) cards. *See* class responsibility collaboration (CRC) cards
- critical path, 354
- CRUD (create, read, update, and delete), OL-48
 - technique, 146–147
- customer relationship management (CRM) system, defined, OL-7
- customers. *See* clients (customers)
- dashboard, project, 346–347
- data, 260–261
- data access classes, 377
- data access layer
 - designing, 419–424
 - linking to domain layer, 419–422
 - relationship with other layers, 200
 - responsibilities of, 427
- data administrator (DA), 262
- data centers, 189, 281–284
- data dictionary, defined, OL-45
- data element definitions, OL-45
- data encryption, 175–177
- data entities, defined, 94, 100
- data entry, 239, 241, 243–244
- data flow, defined, OL-22, OL-43–OL-45
- data flow diagrams (DFD), 62, 137, 161, OL-21–OL-38
 - and abstraction levels, OL-24–OL-28
 - complex information, minimizing, OL-34–OL-35
 - components, documentation of, OL-38–OL-46
 - context diagram, OL-26
 - data element definitions, OL-45
 - data flow consistency, OL-35–OL-38
 - data flow definitions, OL-43–OL-45
 - data store definitions, OL-45
 - decomposition, activity detail, OL-30–OL-32
 - defined, OL-22
 - evaluating quality of, OL-34–OL-38
 - event-partitioned system model, OL-27–OL-28
 - fragments, OL-26–OL-27
 - logical system model, OL-32–OL-34
 - physical system model, OL-32–OL-34
 - process descriptions, OL-38–OL-43
 - RMO, OL-28–OL-32
 - symbols, OL-22–OL-23
 - traditional analysis model, OL-45–OL-46
- data layer. *See* data access layer
- data store, defined, OL-22, OL-45
- data types, 278
- data validation controls, 171
- database administrator (DBA), 262
- database, defined, OL-10
- database lock, 286
- database management system (DBMS), 258–260
- database synchronization, 281
- databases and design. *See also* relational databases
 - administration, 260–263
 - definitions and overview, 22, 167–168, 258
 - distributed architectures, 279–284
 - Downslope Ski Company, 258
 - integrity controls, 170–173
 - key question for, 163
 - project plan and schedule, 261–262
 - protecting, 284–286
 - concurrency and complex update controls, 285–286
 - transaction logging, 285
 - reloading, 455
 - reusing existing, 454
 - for RMO CSMS, 473
 - RMO distributed architecture, 283–284
 - for RMO Supplier Information Subsystem, 22
 - security controls, 173–179
 - team, 262–263
 - technology environment, 260–261
- DBA (database administrator). *See* database administrator (DBA)
- DBMS (database management system), 258–260. *See also* databases and design
- DCDs (design class diagrams). *See* design class diagrams (DCDs)
- decentralized database, 279
- decision table, defined, OL-40
- decision tree, defined, OL-40
- decryption, 176
- dependency relationships, 425
- deployment
 - data conversion and initialization, 454–455
 - direct, 464
 - internal, 200
 - parallel, 464–466
 - phased, 466
 - planning, 464–466
 - production environment configuration, 459–460
 - for RMO CSMS, 473
 - support activities after, 467
 - Tri-State Heating Oil, 444
 - user training, 456–459
- deployment diagrams, 162, 166, 202–203
- design. *See* systems design
- design class diagrams (DCDs)
 - analysis and design models, 162
 - class notation, 377–379
 - defined, 23–24, 372–373
 - for *Fill Shopping Cart*, 413
 - first-cut, 379–382
 - for RMO CSMS, 169
 - for RMO Customer Account Subsystem, 404
 - for RMO Supplier Information Subsystem, 23–24
 - stereotypes in, 376
 - updating, based on sequence diagrams, 424–427
- design constraints, defined, 46
- design patterns. *See also* multilayer design
 - adapter, 428–430
 - controller, use case, 428
 - overview, 427–428
 - singleton, 431–433
- desktop and laptop user interfaces, 238–240
- desktop metaphor, 221
- desktop systems, 239, 261, 376
- destination states, defined, 116
- detailed design, 23, 25–26, 262, 389, 398–399. *See also* object-oriented design/programming (OOD/OOP)
- detailed reports, 246
- detailed sequence diagrams. *See also* design class diagrams (DCDs)
 - for *Create customer account*, 409–410
 - defined, 399
 - for *Fill Shopping Cart*, 410–416, 420, 423
 - guidelines and assumptions for first-cut, 416–417
 - in object-oriented approach, 162, 369, 373
- detailed work schedule, 350
- DFD. *See* data flow diagrams (DFD) diagrams. *See specific diagrams*
- dialogs
 - closure, 228–229
 - developing, 44
 - documenting, 235–237
 - make actions reversible, 229
 - use cases and, 232–234
- dialogue metaphor, 222
- digital certificates, 178
- digital signatures, 178
- direct deployment, 464
- direct manipulation metaphor, 221–222
- disciplines in UP, 309–312
- discount factor, defined, OL-55
- discount rate, defined, OL-55
- distributed database architectures, 279–284
 - implementation approaches for, 280–282
 - RMO, 283–284
- document metaphor, 221–222
- documentation
 - classifications, 456–459
 - establishing environment for, 345–350
 - for RMO CSMS, 473
- domain classes
 - defined, 103
 - for RMO Tradeshow System, 17–18
- domain layer
 - linking to data layer, 419–422
 - relationship with other layers, 199–200
 - responsibilities of, 427
- domain model class diagrams
 - definitions, 103
 - design model based on, 162
 - generalization/specialization relationships, 107–110
 - notation, 104–107
 - relationship with other models, 148
 - for RMO CSMS, 111–114
 - whole-part relationships, 110–111
- domain modeling. *See also* domain model class diagrams; problem domain
 - ERDs, 100–103
 - Waiters on Call, 70, 94
- Downslope Ski Company, 258
- drill down, 249
- drivers (for unit testing), 447
- early start time, defined, OL-60
- EBPs (elementary business processes), 74
- elaboration phase, 309
- electronic reports, 248–250
- Electronics Unlimited, 132
- elementary business processes (EBPs), 74
- embedded software, 192–193
- encryption, 176
- end users, training of, 456–459
- enterprise resource planning (ERP), defined, OL-9
- entity classes, 376
- entity-relationship diagrams (ERDs), 100–103
- environment
 - DBMS, 260–261
 - describing, 203–207
 - design key question, 163
 - design overview, 164
 - establishing, 345–350
 - external, 75, 410
 - key questions, 204–205
 - RMO, 205–207
 - RMO's location diagram, 201–202
 - technology, 260–261
 - work, 349–350
- ERDs (entity-relationship diagrams), 100–103
- errors
 - messages, provide solution options, 231
 - reporting, 468–470
- estimated cost, 340
- estimated time for project completion, 339–340
- event decomposition technique
 - event types, 76–77
 - identifying events, 77–80
 - overview, 74–76
 - steps in, 80
- event-partitioned system model (diagram 0), defined, OL-27

- exception reports, 246
- exclusive lock, 286
- executive reports, 246
- executive stakeholders, 48
- Extensible Markup Language (XML), 194–195
- external agent, defined, OL-22
- external events, 76
- external stakeholders, 47
- Extreme Programming (XP), 312–316
 - core values, 312–313
 - practices, 313–315
 - project activities, 315–316
- Facebook, 41, 46, 48, 188, 190, 196, 206, 285
- factory pattern, 430, 432
- feasibility analysis, 343–344
- feedback, 223–224, 312–313
- field combination controls, 171
- first normal form (1NF), 273–274
- following up interviews, 54
- fonts, 159, 225, 229, 239
- foreign keys
 - normalization and, 273
 - overview, 264
 - representing associations using, 268–270
- fraud prevention, 172–173
- functional decomposition, defined, OL-7
- functional dependency, 274–275
- functional requirements, 45. *See also* use cases
- functional testing, defined, 26
- FURPS and FURPS+, 45–46
- Gang of Four (GoF), 428. *See also* design patterns
- Gantt chart, 353–354, OL-62–OL-69
 - new project, set up, OL-63
 - resources, enter information about, OL-63–OL-64
 - tasks, enter information about, OL-64–OL-69
 - tracking, OL-69
- generalization relationships, 107–110
- GoF (Gang of Four), 428. *See also* design patterns
- graphical models, defined, 59
- graphical presentation, 250
- guard-conditions, defined, 117
- handheld devices. *See* smartphones
- hardware architectures, 261
- HCI (human-computer interaction), 221–222
- heterogeneous distributed database, 279
- high cohesion, 392
- high-level systems design
 - defined, 22
 - for RMO Supplier Information Subsystem, 22–25
- HIOs (human-interface objects). *See* human-interface objects (HIOs)
- homogeneous distributed database, 279
- hosting, 188, 210, 340
 - external, 207
- hours of work in XP, 315
- HTML (Hypertext Markup Language), 194
- HTML5, 195, 206
- HTTP (Hypertext Transfer Protocol), 195
- HTTPS (Hypertext Transfer Protocol, Secure), 179, 195
- human resource management (HRM) systems, 331, OL-8
- human-computer interaction (HCI), 221–222
- human-interface objects (HIOs), 223–224
 - purpose or behavior (affordance), 223
 - visual feedback, 223–224
- hyperlink, 191, 458
- Hypertext Markup Language (HTML), 194
- Hypertext Transfer Protocol (HTTP), 195
- Hypertext Transfer Protocol, Secure (HTTPS), 179, 195
- identifiers (key), defined, 98
- IDEs (integrated development environments), 303
- immediate cutover, 464
- implementation requirements, defined, 46
- inception phase, 308–309
- <<includes>> relationships, 85–87
- incremental development, 300
- indirection, 391
- information gathering
 - brainstorming technique, 95–96
 - collecting active user comments and suggestions, 57–58
 - interviewing users and stakeholders, 50–54
 - observing and documenting business processes, 56–57
 - overview, 43
 - questionnaires, 54
 - researching vendor solutions, 57
 - reviewing inputs, outputs, procedures, 55–56
- information overload, defined, OL-35
- information system development projects, defined, 297
- information systems, OL-6–OL-7. *See also* RMO CSMS (Consolidated Sales and Marketing System)
 - defined, 4, OL-6
 - types of, OL-7–OL-10
- information systems development processes, defined, 6. *See also* Agile development
- inheritance, 108
- input, process, output (IPO) development order, 460–461
- input controls, 171
- input forms, 234–235
- inputs. *See also* system interfaces; user interfaces (UIs)
 - design models, 160–162
 - reviewing, 55–56
 - in SSDs, 139
- instantiation, 368
- intangible benefits, 342
- intangible costs, 343
- integrated development environments (IDEs), 303
- integration testing, 448–450
- integrity controls, 170–173
- interaction diagrams. *See also* detailed sequence diagrams; system sequence diagrams (SSDs)
 - analysis and design models, 162
 - communication diagrams, 399, 400
- interface requirements, defined, 46
- interfaces, overview, 218–219. *See also* system interfaces; user interfaces (UIs)
- internal deployment, 200
- internal stakeholders, 47
- Internet, 190–191
- Internet backbone network, 190
- interoperability, 201
- interviewing users and stakeholders, 50–54
- invented keys, 267–268
- IP Security (IPSec), 179
- iPhone app, 4
- IPO (input, process, output) development order, 460–461
- IPSec (IP Security), 179
 - issues, tracking, 356
- iteration schedule, 15, 353
- iterative development. *See also* Agile development; project management
 - in adaptive SDLC, 299–301
 - overview, 8–9, 27
 - revised plan for RMO CSMS, 472
 - user-centered design and, 220–221
- Javascript, 165
- keys, defined, 98, 264. *See also* foreign keys; primary keys
- knowledge management system (KMS), OL-9
- LANs (local area networks), 190
 - laptop user interfaces, desktop and, 238–240
 - late start time, defined, OL-61
 - level of formality or ceremony, 312, 330
 - levels of abstraction, defined, OL-24
 - lifelines, 140, 409–410
 - list boxes, 239
 - local area networks (LANs), 190
 - location diagrams, 201–202, OL-47
 - location information, networks, OL-47–OL-49
 - logging transactions, 285
 - logical DFD, defined, OL-32
 - loop frames, defined, 142
- maintenance releases, 468
- management. *See* project management
- manufacturing management system, OL-9
- many-to-many associations, 269–270
- mathematical models, defined, 59
- menus, use cases and, 232–234
- metaphors, 221–222, 315
- method signatures, 378
- methodologies, Agile. *See under* Agile development methods
 - class-level, 378
 - deriving from sequence diagrams, 409–410
 - top-down development and, 462
- Microsoft Project, 351–354
- minimization of interfaces, defined, OL-35
- miracle, defined, OL-37
- mobile devices. *See* smartphones
- models and modeling. *See also* domain modeling; object-oriented requirements models
 - agile, 305–307
 - analysis vs. design, 159, 160, 162
 - benefits of, 58
 - defined, 58
 - methodologies and, 301
 - object-oriented approach to, 369–374
 - types overview, 59
- Mountain Vista Motorcycles, 38
- multifactor authentication, 174
- multilayer design
 - data access layer, 419–424
 - developing, 417–424
 - domain layer. *See* use case realization
 - implementation issues for, 426–427
 - object-oriented design with interaction diagrams, 399
 - responsibilities of each layer, 427
 - top-down development of, 462
 - view layer, 417–418
- multimedia presentation, 250
- multiplicity
 - defined, 99
 - UML notation for, 104
- multiplicity constraints, defined, 99
- n-ary associations, defined, 100
- navigation controls, 243, 245
- navigation visibility, 380–381, 416
- net present value (NPV), 341, OL-55–OL-57
- network architectures, 261
- network-based systems, 427
- network communication, OL-47–OL-49
- network diagrams, 202, 205, OL-69–OL-70
- network protocols, 193–194
- network-based systems, 427
- networks, 190–191
- New Capital Bank, 366, 398
- New Mexico Health Systems, security and system controls, 158–159
- nonfunctional requirements, defined, 45
- normalization, database, 272–278
- Norman, Donald, 221
- noun technique, 96–97
- NPV (net present value). *See* net present value (NPV)
- object lifelines, 140
- object responsibility, 389–390
- object-oriented approach, 103, 108, 139, 367, 378. *See also* object-oriented design/programming (OOD/OOP)
 - system development, OL-21
- object-oriented design/programming (OOD/OOP). *See also* use case realization
 - analysis to implementation, 367–374
 - with communication diagrams, 403–408
 - extend input messages, 404–407
 - final design class diagram, 407–408
 - input models, 404
 - with CRC cards, 382–388
 - defined, 374
 - fundamental principles, 388–392
 - with interaction diagrams, 399–401
 - models, 369–374
 - New Capital Bank, 366, 398
 - overview, 367–369, 388–389
 - steps, 374–375
- object-oriented requirements models. *See also* system sequence diagrams (SSDs); state machine diagrams

- activity diagrams for use cases, 137–139
 - Electronics Unlimited, 132
 - integrating, 148
 - overview, 132–133
 - use case descriptions, 133–136
- objects, defined, 103, 114
- one-to-many associations, 268
- OOD/OOP (object-oriented design/programming). *See* object-oriented design/programming (OOD/OOP)
- open items, tracking, 54
- open-ended questions, 51
- operational stakeholders, 47–48
- opt frames, 142, 143
- order of development, 460–463, 473
- organizational risks and feasibility, 343
- origin states, defined, 116
- output controls, 171–172
- outputs. *See also* reports; system interfaces; user interfaces (UIs)
 - design models, 160–162
 - reviewing, 55–56
 - in SSDs, 139
- outstanding items, tracking, 54
- oversight committees, 329, 330
- ownership of code in XP, 314
- package diagrams
 - analysis and design models, 162
 - for RMO, 461
 - structuring major components with, 424–426
- pair programming, 314
- parallel deployment, 464–466
- partitioned database server architecture, 283
- paths, defined, 117
- patterns, defined, 427. *See also* design patterns
- payback period, 342, OL-57
- people knowledge, for systems analyst, OL-12–OL-13
- perfect internal technology, defined, OL-32
- perfect memory assumption, 417
- perfect solution assumption, 417
- perfect technology assumption, 79, 417
- performance and throughput requirements, defined, 46
- testing, 451
- persistent classes, 376, 419
- PERT charts, OL-58–OL-62, OL-69–OL-70
- phased deployment, 466
- phases
 - project, 298–299
 - UP, 308–309
- phones. *See* smartphones
- physical data store, 176, 259, 260
- physical DFD, defined, OL-32
- physical requirements, defined, 46
- pie chart sample, 250
- Pinnacle Manufacturing, 296
- planning. *See* project management
- planning in XP, 313
- plug-ins, 192
- PMBOK (Project Management Body of Knowledge). *See* Project Management Body of Knowledge (PMBOK)
- postconditions, 135
- preconditions, 135
- predictive approaches to the SDLC, 297, 298–299, 301. *See also* project management
- primary keys
 - choosing, 266–268
 - normalization and, 272–278
 - overview, 264
- primitive data types, 278
- printers and output controls, 171
- prioritizing requirements, 44
- privileged users, 175
- problem domain
 - associations among things in, 98–100
 - attributes of things in, 97–98
 - brainstorming technique, 95–96
 - defined, 94
 - noun technique, 96–97
- problem identification, 336–337
- procedures, reviewing, 55–56
- process, defined, OL-22
- process descriptions, DFD, OL-38–OL-43
- processes, business. *See* business processes
- procurement management, 331
- product backlog, 317
- product owner, 317
- production environment configuration, 459–460
- production releases, 468
- production system, 471
- production versions, 468
- programming for RMO Supplier Information Subsystem, 25–26
- progress monitoring, 355–356
- project communications management, OL-74–OL-75
 - objectives of, OL-74–OL-75
 - techniques, OL-75
- project cost management, OL-73
 - objectives of, OL-73
 - techniques, OL-73
- project dashboard, 346–347
- project human resources management, OL-74
 - objectives of, OL-74
 - techniques, OL-74
- project integration management, OL-77
- project iteration schedule, 350
- project management. *See also* System Development Life Cycle (SDLC)
 - agile, 332–335
 - Blue Sky Mutual Funds, 326
 - body of knowledge for, 331–332
 - ceremony and, 330–331
 - change and version control, 467–471
 - defined, 329
 - deployment planning, 463–466
 - determining project risk and feasibility, 343–344
 - development order, 460
 - establishing environment, 345–350
 - evaluating work processes, 354–355
 - identifying problem, 336–337
 - monitoring progress and making corrections, 355–356
 - need for, 327–328
 - overview, 327
 - putting it all together, for RMO, 471–473
 - quantifying approval factors, 337–343
 - reviewing with client and obtaining approval, 344–345
 - role of manager, 329–330
 - scheduling work, 350–354
 - source code control, 462–463
 - staffing and allocating resources, 354
- Project Management Body of Knowledge (PMBOK), OL-54, OL-70–OL-71
 - project communications management, OL-74–OL-75
 - project cost management, OL-73
 - project human resources management, OL-74
 - project integration management, OL-77
 - project procurement management, OL-75–OL-76
 - project quality management, OL-73–OL-74
 - project risk management, OL-75
 - project scope management, OL-71–OL-72
 - project stakeholder management, OL-76
 - project time management, OL-72–OL-73
- project management techniques
 - Gantt chart, OL-62–OL-69
 - net present value calculations, OL-55–OL-57
 - payback period calculations, OL-57
 - PERT/CPM charts, understanding, OL-58–OL-62, OL-69–OL-70
 - project schedule building, microsoft project, OL-62–OL-69
 - return on investment, OL-57–OL-58
- project procurement management, OL-75–OL-76
 - objectives of, OL-75–OL-76
 - techniques, OL-76
- project quality management, OL-73–OL-74
 - objectives of, OL-73
 - techniques, OL-74
- project risk management, OL-75
 - objectives of, OL-75
 - techniques, OL-75
- project scope management, OL-71–OL-72
 - objectives of, OL-71–OL-72
 - techniques, OL-72
- project stakeholder management, OL-76
 - objectives of, OL-76
 - techniques, OL-76
- projects. *See also* project management; RMO entries; System Development Life Cycle (SDLC)
 - defined, 7, 297
 - phases, 298
 - reasons for initiating, 336
 - stakeholders in, 330
- project time management, OL-72–OL-73
 - objectives of, OL-72
 - techniques, OL-72–OL-73
- protection from variations, 390–391
- protocols
 - defined, 193
 - network, 193–194
 - Web, 194–195
- pseudostates, defined, 116
- public key encryption, 177
- quality management, 331, 335
- questionnaires, 54
- questions, interview, 51
- RAD (rapid application development), 331
- radio buttons, 224
- rapid application development (RAD), 331
- RDBMS, 263. *See also* relational databases
- read lock, 286
- recording information, environment for, 346–349
- recovery procedures, 172
- recursive associations, defined, 100
- redundancy, 172
- refactoring, 314
- referential integrity, enforcing, 271–272
- registered users, 174–175
- relational databases
 - associations, 268–270
 - classification hierarchies, 270–271
 - data types, 278
 - definitions and overview, 263
 - design overview, 264–265
 - normalization, 272–278
 - primary keys, 266–268
 - referential integrity, 271–272
 - table creation, 265
- relationships. *See also* associations
 - defined, 99
 - generalization/specialization, 107–110
 - whole-part, 110–111
- release versions, 468
- releases in XP, 315–316
- reliability requirements, defined, 45
- remote wipe, 177
- replicated database server architecture, 280–281
- reports
 - electronic, 248–250
 - graphical and multimedia presentation, 250
 - output controls, 171–172
 - types of, 246–248
- requirements. *See* object-oriented requirements models; system requirements
- resource allocation, 354
- resource risks and feasibility, 344
- response time, 451
- responsibilities
 - on CRC cards, 382–388
 - object, 389–390
 - separation of, 419
- retrospectives, 354
- return on investment (ROI), OL-57–OL-58
- reversible action (undo), 229
- Ridgeline Mountain Outfitters (RMO). *See also* RMO entries
 - data flow diagrams, OL-28–OL-32
- risk
 - analysis, 343–344
 - management, 331, 335
- RMO CSMS (Consolidated Sales and Marketing System). *See also other* RMO entries
 - annual operating costs for, 340
 - anticipated benefits from, 340–343
 - application architectures, 208–210
 - associations among things in, 99
 - database architecture and plan for, 283–284, 473

- deployment plan for, 472–473
- development order for, 473
- documentation and training, 473
- domain model class diagram for, 111–114, 266
- estimated costs for developing, 340
- existing system and architecture, 39–40, 201–202
- information repositories for, 347–348
- iteration plan, revised, 472
- menu design based on use cases, 234
- order fulfillment activity diagram for, 61
- overview, 41
- package diagram for, 461
- problem domain for, 95
- putting it all together, 471–473
- samples
 - associations, 99
 - CRC card, 382
 - database table definition, 168
 - design class diagram, 169
 - external events, 80
 - iteration schedule, 353
 - mail-order form, 56
 - open-items list, 54
 - product detail search screen, 240
 - questionnaire, 55
 - reports, 246, 247, 249
 - sales types, 108
 - things in problem domain, 95, 96–97
 - time estimate document, 339–340
 - use case diagrams, 81–85
 - use cases, 80–87, 146–147
 - user goals, 73–74
 - Web pages, 231
- stakeholders for, 48–49
- state machine diagrams for, 119–122
- “upgrade or replace” decision, 471
- RMO Customer Account Subsystem
 - domain model class diagram for, 113–114
 - package diagram for, 461
- RMO environment description, 205–207
 - external hosting, 207
 - mobile devices and apps, 206
 - security implications, 206–207
 - social networking, 206
 - updated RMO technology architecture, 207
 - Web technologies and adapted content, 206
- RMO Product Information Subsystem
 - class diagrams for, 17–18
 - use cases for, 16–17
- RMO Sales Subsystem
 - domain model class diagram for, 111–113, 379
 - package diagram for, 461
 - samples
 - CRC cards, 388
 - design class diagrams, 382, 389
 - project iteration schedule, 350
 - work breakdown structure, 351–352
- RMO Supplier Information Subsystem
 - architecture, 24–25
 - class diagrams for, 17–18
 - database design, 22
 - domain classes for, 17–18
 - fact finding and user involvement, 16
 - general approach design, 22–25
 - planning rest of first iteration, 14–16
 - programming, 25–27
 - screen layout, 20–22
 - testing, 26–27
 - use cases for, 17–19
 - workflow diagrams development, 19, 20
- RMO, three-layer package diagram for, 426
- RMO Tradeshow System. *See also other RMO entries*
 - initial project activities, 11–12
 - introduction, 9–11
 - managing project, 25
 - planning overall project and iterations, 12–14
 - software components, 23
 - subsystems of, 12–13
 - System Vision Document, 12
- ROI (return on investment). *See return on investment (ROI)*
- rows, database, 263
- rule of 7 ± 2 (Miller’s number), defined, OL-35
- SAAS (software as a service), 196
- SCCS (source code control system), 462–463
- scenarios, defined, 134
- schedule risks and feasibility, 344
- schema, database, 259
- scope, 331, 333, 337
- Scrum
 - overview, 316
 - philosophy, 316–318
- Scrum master, 317–318
- SDLC (System Development Life Cycle). *See System Development Life Cycle (SDLC)*
- second normal form (2NF), 275–276
- Secure Sockets Layer (SSL), 179
- security and system controls
 - for databases, 173–179
 - design key question, 163
 - events involving, 79–80
 - integrity controls, designing, 170–173
 - backup, 172
 - fraud prevention, 172–173
 - input controls, 171
 - output controls, 171–172
 - recovery, 172
 - redundancy, 172
 - scenarios, 170
 - at New Mexico Health Systems, 158–159
 - overview, 168–169
 - security controls, designing, 173–179
 - access controls, 174–175
 - data encryption, 175–177
 - digital signatures and certificates, 177–179
 - secure transactions, 179
 - in VPNs, 194
 - security requirements, defined, 46
 - semantic nets, 102
 - separation of responsibilities, 419
- sequence diagrams. *See detailed sequence diagrams; system sequence diagrams (SSDs)*
- server computers, defined, 188, 199
- server farms, 199, 261
- shared lock, 286
- shortcuts, 231
- singleton pattern, 431–433
- slack time, defined, OL-62
- smartphones, 12, 23, 206, 210, 238, 298
- user interface guidelines, 242–244
- software
 - application components
 - boundaries, 208
 - integration, 210–213
 - RMO CSMS application architecture, 208–210
 - classes and methods, designing, 168
 - components, Tradeshow System, 23
 - embedded, 192–193
 - overview, 191
 - Web-based applications, 191–192
- software as a service (SAAS), 196
- source code control system (SCCS), 462–463
- specialization relationships, 107–110
- sprints, 318
- SQL (Structured Query Language), 259, 263
- SSDs (system sequence diagrams). *See system sequence diagrams (SSDs)*
- SSL (Secure Sockets Layer), 179
- staffing, 354
- stakeholders. *See also clients (customers); information gathering; users*
 - categories of, 47–48
 - defined, 47
 - internal and external, 330
 - for RMO CSMS, 48–49
- stand-alone software application, 238
- Standish Group, 328
- state events, 77
- state machine diagrams
 - concurrency and concurrent states, 117–118
 - definitions, 114–117
 - developing, 118–119
 - in OOD, 162
 - relationship with other models, 148
 - for RMO, 119–122
- status, collecting and reporting, 355–356
- steering committees, 329, 348
- stereotypes, 376–377, 433
- storyboarding, 236, 237
- stress testing, 451
- structured English, defined, OL-38
- Structured Query Language (SQL), 259, 263
- stubs, 447
- subclasses, defined, 107
- subsystems, defined, 12–13, OL-6
- summary reports, 246
- superclasses, defined, 107
- supply chain management (SCM) system, OL-7–OL-8
- support
 - activities, 467
 - staff, as stakeholders, 48
- supportability requirements, defined, 46
- swimlane, defined, 60
- symmetric key encryption, 176
- synchronization, database, 281
- synchronization bars, defined, 60
- system architectures
 - concepts, 195–200
 - modern system, anatomy of, 187–195
 - computing devices, 188–189
 - Internet, 190–191
 - networks, 190–191
 - protocols, 193–195
 - software, 191–193
 - World Wide Web, 190–191
 - overview, 186–187
- system boundary, defined, OL-7
- system capabilities, defined, 337
- system controls. *See security and system controls system, defined, OL-6*
- System Development Life Cycle (SDLC). *See also Agile development; object-oriented design/programming (OOD/OOP)*
 - adaptive approaches, 298, 299–301
 - change implementation, 470–471
 - core processes, 7. *See also iterative development methodologies, models, tool, techniques, 301–304*
 - overview, 7–8, 297
 - predictive approaches, 297, 298–299, 301
 - sample cases, 296. *See also RMO Tradeshow System*
 - support activities after deployment, 467
- system development process (methodology), 7
- system documentation, 456–457
- system interfaces, 29, 377
- system metaphor, 315
- system of record, 212
- system operators, training of, 456–459
- system requirements. *See also information gathering; models and modeling; stakeholders*
 - categories of, 45–46
 - defined, 45
 - defining (activity), 43
 - documenting workflows, 60–63
 - evaluating with users, 44
 - prioritizing, 44
 - sample cases
 - Mountain Vista Motorcycles, 38
 - Ridgeline Mountain Outfitters. *See RMO CSMS (Consolidated Sales and Marketing System)*
- system sequence diagrams (SSD)
 - defined, 139
 - design model based on, 162, 409–410
 - developing, 142–146
 - for *Fill Shopping Cart*, 412
 - notation, 140–142
 - relationship with other models, 148
- system software, 191
- system testing, 450
- System Vision Documents
 - components of, 337
 - for RMO CSMS, 337
 - for RMO Tradeshow System, 12
- systems analysis, OL-3. *See also object-oriented requirements models*
 - activities overview, 42–44
 - defined, 4, 42–43
 - design and implementation, 160, 161
 - importance of, 4–6

- models in, 161
 - systems design, 160
 - transition to user-interface design, 232–237
 - systems analysts
 - analysis-related careers and, OL-13–OL-15
 - business knowledge for, OL-12
 - as business problem solver, OL-3–OL-7
 - at consolidated refineries, OL-2
 - defined, 6, OL-3
 - integrity and ethics, OL-13
 - people knowledge for, OL-12–OL-13
 - skills of, OL-10–OL-13
 - technical knowledge for, OL-10–OL-11
 - systems controls, designing, 163
 - systems design
 - activities overview, 163–168
 - defined, 5, 159–160, OL-3
 - design and implementation, 160, 161
 - design models, 160–162
 - importance of, 4–6
 - overview, 159–160
 - systems analysis, 160
 - systems development methodologies, 301
 - tables, database, 263
 - tablets, user-interface guidelines, 244–245
 - tangible benefits, 342
 - teams in Agile projects, 354
 - technical knowledge, for systems analyst, OL-10–OL-11
 - technical staff as stakeholders, 48
 - techniques, 303, OL-11
 - CRUD, 146–147
 - event decomposition, 75–76
 - information-gathering, 50–58
 - methodologies and, 301–304
 - noun, 96–97
 - use cases and user goal, 73–74
 - technological risks and feasibility, 343–344
 - technology architecture, 39
 - templates, defined, 427. *See also* design patterns
 - temporal events, 77
 - ternary associations, defined, 100
 - test versions, 468
 - testing
 - build and smoke, 451
 - definitions and overview, 446–447
 - integration, 448–450
 - order of development and, 460–462
 - overview, 26
 - performance, 451
 - for RMO Supplier Information System, 26–27
 - stress, 451
 - system, 450
 - Tri-State Heating Oil, 444
 - unit, 447–448
 - usability, 468
 - user acceptance, 26, 451–453
 - in XP, 314
 - text boxes, 239
 - textual models, defined, 59
 - themes for information gathering questions, 50–51
 - things in problem domain, 97–100
 - third normal form (3NF), 276–278
 - three-layer client-server architecture
 - environment design, 199–200
 - implementation issues for, 426–427
 - responsibilities of each layer, 427
 - top-down development of, 462
 - use of in internal deployments, 200
 - throughput. *See* performance and throughput
 - time
 - estimated, 339–340
 - management of, 331, 333–334
 - TLS (Transport Layer Security), 179
 - toolbars, 192
 - tools
 - defined, OL-11
 - methodologies and, 301–304
 - top-down development, 462
 - top-down programming, 462
 - tracking logs, 356
 - Traditional analysis model, DFD, OL-45–OL-46
 - traditional approaches
 - predictive, 297, 298–299, 301
 - system development, OL-21
 - training, 456–459, 473
 - transaction logging, 285
 - transition phase, 308–309
 - transitions, defined, 114
 - Transport Layer Security (TLS), 179
 - Tri-State Heating Oil, 444
 - true/false conditions, defined, 142
 - turnaround documents, 245–250
 - UATs (user acceptance tests). *See* user acceptance tests (UATs)
 - UIs. *See* user interfaces (UIs)
 - UML. *See* Unified Modeling Language (UML)
 - unary associations, defined, 100
 - unauthorized users, 174
 - undoing actions, 229
 - Unified Modeling Language (UML). *See also* activity diagrams; class diagrams; detailed sequence diagrams; system sequence diagrams (SSDs)
 - component diagrams, 162
 - defined, 59
 - stereotyping in, 87, 376
 - Unified Process (UP), 308–312, 331
 - disciplines, 309–312
 - phases, 308–309
 - Uniform Resource Locator (URL), 190
 - unit testing, 447–448
 - unresolved issues, tracking, 54
 - UP disciplines, 309–312
 - UP (Unified Process). *See* Unified Process (UP)
 - URL (uniform resource locator), 190
 - usability
 - requirements, 45
 - testing, 451
 - as user-centered design principle, 221
 - Use Case Controller, 400–401
 - use case descriptions
 - design model based on, 162
 - main discussion, 133–136
 - relationship with other models, 148
 - use case diagrams
 - defined, 81
 - design model based on, 162
 - developing, 87
 - including other use cases in, 85–87
 - relationship with other models, 148
 - for RMO CSMS, 81–87
 - for RMO Supplier Information Subsystem, 18–19
 - use case instances, defined, 134
 - use case realization. *See also* multilayer design
 - with communication diagrams, 401–408
 - design process for, 410
 - overview, 398
 - with sequence diagrams for *Fill Shopping Cart*, 410–416
 - updating and packaging design classes, 424–427
 - use-case-driven development, 462
 - use cases. *See also* design class diagrams (DCDs); event decomposition technique; use case diagrams; use case realization
 - activity diagrams for, 137–139
 - controllers, 400–401, 428
 - CRUD technique, 146–147
 - defined, 16, 73
 - descriptions, 133–136
 - order of development and, 462
 - for RMO CSMS, 80–81, 146–147
 - for RMO Tradeshow System, 16
 - user goal technique, 73–74
 - user stories and, 71–73
 - user-interface design and, 232–237
 - Walters On Call, 70
 - user acceptance tests (UATs), 26, 316, 451–453
 - management and execution, 453
 - planning, 452
 - preparation and pre-activities, 452–453
 - user documentation, 456–459
 - user experience (UX)
 - defined, 219
 - understanding, 219–222
 - user goal technique, 73–74
 - user interfaces (UIs). *See also* dialogs
 - adding CRC cards for, 383, 385
 - analysis models and input forms, 234–235
 - defined, 219
 - design key question, 163
 - desktop and laptop, 238–240
 - fundamental principles, 223–232
 - closure, 228–229
 - consistency, 224–226
 - discoverability, 226–228
 - HIOs, 223–224
 - readability and navigation, 229–230
 - usability and efficiency, 230–232
 - menus, 232–234
 - metaphors for human-computer interaction, 221–222
 - order of development and, 461–462
 - overview, 165–167, 237–238
 - principles, 219–221
 - for RMO Supplier Information Subsystem, 19, 24–25
 - smartphones and small mobile devices, 242–244
 - tablets, 244–245
 - transition from analysis to design of, 232–237
 - universal guidelines, 223–232
 - Web-based applications, 240–242
- user stories, 316
 - with acceptance criteria, 72
 - defined, 71
 - and use cases, 71–73
- user-centered design principles, 220–221
- users. *See also* information gathering; stakeholders
 - access control for, 174–175
 - defined, 329
 - evaluating requirements with, 44
 - gathering information from, 43
 - involvement of, 16
 - in Scrum, 318
 - training of, 456–459
 - in XP, 314
- UX (user experience). *See* user experience (UX)
- value limit controls, 171
- variance, analyzing, 355
- variations, protection from, 390–391
- vendors, researching solutions from, 57
- version control, 467–471
- videos on Web pages, 190
- view classes, 376–377
- view layer
 - designing, 417–419
 - relationship with other layers, 199–200
 - responsibilities of, 427
- Virtual Private Network (VPN), 194
- virtual servers, 349
- visibility, 223–224, 377
- visibility, navigation, 380–382, 416
- visual modeling tools, 303
- VPN (Virtual Private Network), 194
- Walters On Call, 70, 94
- walking skeletons, 300–301
- waterfall model, 299
- WBS (work breakdown structure), 14, 351–352
- Web protocols, 194–195
- Web service, 197–198
 - data import via, 211
- Web technologies. *See* browser-based systems; Internet
- Web-based applications, 191–192
 - user interface guidelines for, 240–242
- whole-part relationships, 110–111
- widgets, 192
- work
 - environment, establishing, 349–350
 - processes, evaluating, 354–355
 - scheduling, 350–354
- work breakdown structure (WBS), 14, 351–352
- work hours in XP, 315
- workflow diagrams, 19–20
- workflows, documenting, 60–63
- World Wide Web (WWW), 190–191, 194
- write lock, 286
- WW (World Wide Web), 190–191, 194
- Wysotronics Inc., 186
- XML (Extensible Markup Language), 194–195
- XP (Extreme Programming). *See* Extreme Programming (XP)