

Systems Analysis and Design with UML Version 2.0

AN OBJECT-ORIENTED APPROACH

Second Edition

Alan Dennis
Indiana University

Barbara Haley Wixom
University of Virginia

David Tegarden
Virginia Tech



John Wiley & Sons, Inc.
www.wiley.com/college/dennis

Acquisitions Editor	<i>Beth Lang Golub</i>
Associate Editor	<i>Lorrain Raccuia</i>
Editorial Assistant	<i>Ame Esterline</i>
Marketing Manager	<i>Jillian Rice</i>
Media Editor	<i>Allie K. Morris</i>
Production Manager	<i>Pam Kennedy</i>
Production Editors	<i>Kelly Tavares, Sarah Wolfman-Robichaud</i>
Managing Editor	<i>Kevin Dodds</i>
Illustration Editor	<i>Benjamin Reece</i>
Cover Design	<i>Benjamin Reece</i>
Cover Image	© Digital Vision/Getty Images

This book was set in Minion by Leyh Publishing, LLC and printed and bound by RR Donnelley–Willard. The cover was printed by Phoenix Color Corp.

This book is printed on acid free paper. ☺

Copyright © 2005 John Wiley & Sons, Inc. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc. 222 Rosewood Drive, Danvers, MA 01923, (978)750-8400, fax (978)750-4470. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201)748-6011, fax (201)748-6008, E-Mail: PERMREQ@WILEY.COM.

To order books or for customer service please call 1-800-CALL WILEY (225-5945).

ISBN 0-471-34806-6

WIE ISBN 0-471-65920-7

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

PREFACE

PURPOSE OF THIS BOOK

Systems Analysis and Design (SAD) is an exciting, active field in which analysts continually learn new techniques and approaches to develop systems more effectively and efficiently. However there is a core set of skills that all analysts need to know—no matter what approach or methodology is used. All information systems projects move through the four phases of planning, analysis, design, and implementation; all projects require analysts to gather requirements, model the business needs, and create blueprints for how the system should be built; and all projects require an understanding of organizational behavior concepts like change management and team building. Today, the cost of developing modern software is composed primarily of the cost associated with the developers themselves and not the computers. As such, object-oriented approaches to developing information systems hold much promise in controlling these costs.

Today, the most exciting change to systems analysis and design is the move to object-oriented techniques, which view a system as a collection of self-contained objects that have both data and processes. This change has been accelerated through the creation of the Unified Modeling Language (UML). UML provides a common vocabulary of object-oriented terms and diagramming techniques that is rich enough to model any systems development project from analysis through implementation.

This book captures the dynamic aspects of the field by keeping students focused on doing SAD while presenting the core set of skills that we feel every systems analyst needs to know today and in the future. This book builds on our professional experience as systems analysts and on our experience in teaching SAD in the classroom.

This book will be of particular interest to instructors who have students do a major project as part of their course. Each chapter describes one part of the process, provides clear explanations on how to do it, gives a detailed example, and then has exercises for the students to practice. In this way, students can leave the course with experience that will form a rich foundation for further work as a systems analyst.

OUTSTANDING FEATURES

A Focus on Doing SAD

The goal of this book is to enable students to do SAD—not just read about it, but understand the issues so they can actually analyze and design systems. The book introduces each major technique, explains what it is, explains how to do it, presents an example, and provides opportunities for students to practice before they do it for real in a project. After reading each chapter, the student will be able to perform that step in the system development life cycle (SDLC) process.

Rich Examples of Success and Failure

The book includes a running case about a fictitious company called CD Selections. Each chapter shows how the concepts are applied in situations at CD Selections. Unlike running cases in other books we have tried to focus these examples on planning, managing, and executing the activities described in the chapter, rather than on detailed dialogue between fictitious actors. In this way, the running case serves as a template that students can apply to their own work. Each chapter also includes numerous Concepts in Action boxes that describe how real companies succeeded—and failed—in performing the activities in the chapter. Many of these examples are drawn from our own experiences as systems analysts.

Real World Focus

The skills that students learn in a systems analysis and design course should mirror the work that they ultimately will do in real organizations. We have tried to make this book as “real” as possible by building extensively on our experience as professional systems analysts for organizations such as Arthur Andersen, IBM, the U.S. Department of Defense, and the Australian Army. We have also worked with a diverse industry advisory board of IS professionals and consultants in developing the book and have incorporated their stories, feedback, and advice throughout. Many students who use this book will eventually use the skills on the job in a business environment, and we believe they will have a competitive edge in understanding what successful practitioners feel is relevant in the real world.

Project Approach

We have presented the topics in this book in the SDLC order in which an analyst encounters them in a typical project. Although the presentation is necessarily linear (because students have to learn concepts in the way in which they build on each other), we emphasize the iterative, complex nature of SAD as the book unfolds. The presentation of the material should align well with courses that encourage students to work on projects because it presents topics as students need to apply them.

WHAT'S NEW IN THIS EDITION

This new edition has four major improvements. First, we converted the entire text from UML 1.3 to UML 2.0. Due to the size and complexity of UML 2.0, we factored the object-oriented material out of Chapter 1 and created a new chapter (Chapter 2) that includes an introduction to object orientation, UML 2.0, the Unified Process, and which overviews a minimalist approach to Object-Oriented Systems Analysis and Design with UML 2.0. Some of this material originally was included in Chapter 9. However, for continuity and flexibility purposes, we moved this material to Chapter 2. This new chapter is organized in such a way to allow the instructor the maximum amount of flexibility to choose what material they want to emphasize. For example, if the students have already had an object-oriented programming course using Java, the basic characteristics of object-oriented systems section could either be assigned for reading or review only. Or, if the instructor would prefer to minimize the amount of UML to be introduced to the student at this point in the course, the UML 2.0 section could be optional. The details of the relevant UML 2.0 are covered in the book where the diagrams are applicable, e.g., class diagrams are covered in detail

in Chapter 7: Structural Models. In this section of Chapter 2, we only introduce the purpose of the fourteen diagrams that are now included in the UML.

Second, not only have we updated the text to cover UML 2.0, we also have expanded the coverage of UML. In this edition, we have added activity diagrams to support business process modeling (see Chapter 6) and deployment diagrams to provide for modeling the physical architecture of the system (see Chapter 13).

Third, we combined some chapters together to improve the flow of material through the book. For example, in the first edition we had two chapters devoted to user interface design. In this edition we have combined the material into a single chapter.

Fourth, we reordered the material dealing with object-oriented systems design contained in Part Two. The new order is class and method design (Chapter 10), object storage design (Chapter 11), user interface design (Chapter 12), and finally, the design of the physical architecture of the system (Chapter 13). The new order provides a more logical organization of this material.

Of course, one feature that we were sure to keep was to keep the footnotes in the chapters. Even though at first glance footnotes do not seem to add a lot to a textbook, they provide a means for the student and/or instructor to delve deeper into any topic included in this text. With the expanded material, we have added quite a few footnotes throughout the text.

ORGANIZATION OF THIS BOOK

This book is organized by the phases of the Systems Development Life Cycle (SDLC). Each chapter has been written to teach students specific tasks that analysts need to accomplish over the course of a project, and the deliverables that will be produced from the tasks. As students complete the book, tasks will be “checked off” and deliverables will be completed. Along the way, students will be reminded of their progress using roadmaps that indicate where their current task fits into the larger context of SAD.

Chapter 1 introduces the SDLC and describes the roles and skills needed for a project team. Chapter 2 introduces the basic characteristics of object-oriented systems, UML 2.0, Object-Oriented Systems Analysis, and the Unified Process. It also overviews a minimalist approach to Object-Oriented Systems Analysis and Design with UML 2.0.

Part One contains Chapters 3 and 4 which describe the Planning Phase. Chapter 3 presents Project Initiation, with a focus on the System Request, Feasibility Analysis, and Project Selection. In Chapter 4, students learn about Project Management, with emphasis on the Workplan, Staffing Plan, Project Charter, and Risk Assessment that are used to help manage and control the project.

Part Two presents techniques needed during the Analysis Phase. In Chapter 5, students are introduced to a variety of requirements gathering techniques that are used to create an Analysis Plan after learning an assortment of analysis techniques to help with Business Automation, Business Improvement, and Business Process Reengineering. Using the identified requirements, Chapter 6 focuses on constructing Functional Models, Chapter 7 addresses producing Structural Models, and Chapter 8 tackles creating Behavioral Models.

Part Three addresses the Design Phase. In Chapter 9, students learn how to evolve the analysis models into design models via the use of factoring, partitions, and layers. The students also learn to create an Alternative Matrix that can be used to compare custom, packaged, and outsourcing alternatives. Chapter 10 concentrates on designing the individual

classes and their respective methods through the use of contracts and method specifications. Chapter 11 presents the issues involved in designing persistence for objects. These issues include the different storage formats that can be used for object persistence, how to map an object-oriented design into the chosen storage format, and how to design a set of data access and manipulation classes that act as a translator between the classes in the application and the object persistence. Chapter 12 presents the design of the human computer interaction layer where students learn how to design user interfaces using Use Scenarios, Windows Navigation Diagrams, Real Use Cases, Interface Standards, and User Interface Templates. Chapter 13 focuses on the physical architecture design, which includes Deployment Diagrams and Hardware/Software Specification.

Part Four provides material that is related to the Implementation Phase. Chapter 14 focuses on system construction where students learn how to build, test, and document the system. Installation and operations are covered in Chapter 15 where students learn about the Conversion Plan, Change Management Plan, Support Plan, and Project Assessment.

SUPPLEMENTS <http://www.wiley.com/college/dennis>

Instructor's Resources Web Site

- PowerPoint slides, that instructors can tailor to their classroom needs and that students can use to guide their reading and studying activities
- Test Bank, that includes a variety of questions ranging from multiple choice to essay style questions. A computerized version of the Test Bank will also be available.

Online Instructor's Manual

The Instructor's Manual provides resources to support the instructor both inside and out of the classroom:

- Short experiential exercises that instructors can use to help students experience and understand key topics in each chapter.
- Short stories have been provided by people working in both corporate and consulting environments for instructors to insert into lectures to make concepts more colorful and real
- Additional mini-cases for every chapter allow students to perform some of the key concepts that were learned in the chapter.
- Solutions to end of chapter questions and exercises are provided.

Student Web Site

- Relevant Web links, including career resources Web site.
- Web Quizzes help students prepare for class tests.

Cases in Systems Analysis and Design

A separate Case Book on CD-ROM, provides a set of more than a dozen cases that can be used to supplement the book and provide exercises for students to practice with. The cases are primarily drawn from the U.S. and Canada, but also include a number of international cases. We are always looking for new cases, so if you have a case that might be appropriate please contact us directly (or your local Wiley sales representative).

Software Tools

Five Software Tools can be purchased with the text in special packages:

1. Oracle's 9i Database Suite, which includes Oracle 9i Personal Edition, Oracle 9i Enterprise Edition, and Oracle 9i Standard Edition.
2. Oracle's Developer Suite, which includes Oracle Developer and Oracle Designer. This software is available under a "Development Sublicense" for personal development purposes only, and has no time restrictions or limitations.
3. Visible Systems Corporation's Visible Analyst Student Edition.
4. Microsoft's Visio
5. Microsoft's Project

Contact your local Wiley sales representative for details, including pricing and ordering information.

ACKNOWLEDGEMENTS

Thanks to Roberta Roth, University of Northern Iowa and Suresh Chalasani of University of Wisconsin Parkside, for their work on the supplements.

We would like to thank the following reviewers for their helpful and insightful comments on the second edition: Murugan Anandarajon, Drexel University; Ron Anson, Boise State University; Noushin Ashrafi, University of Massachusetts Boston; Dirk Baldwin, University of Wisconsin; Robert Barker, University of Louisville; Terry Fox, Baylor University; Donald Golden, Cleveland State University; Cleotilde Gonzalez, Carnegie Mellon University; Scott James, Saginaw Valley State University; Rajiv Kishore, State University of New York–Buffalo; Ravindra Krovi, University of Akron; Fernando Maymi, United States Military Academy at West Point; Fred Niederman, Saint Louis University; Graham Peace, West Virginia University; J. Drew Procaccino, Rider University; Marcus Rothenberger, University of Wisconsin–Milwaukee; June Verner, Drexel University; Heinz Roland Weistroffer, Virginia Commonwealth University; and Amy Woszcynski, Kennesaw State University.

We also thank the following reviewers from the first edition: Evans Adams, Fort Lewis College; Noushin Ashrafi, University of Massachusetts, Boston; Dirk Baldwin, University of Wisconsin-Parkside; Qing Cao, University of Missouri–Kansas City; Ahmad Ghafarian, North Georgia College & State University; Daniel V. Goulet, University of Wisconsin–Stevens Point; Harvey Hayashi, Loyalist College of Applied Arts and Technology; Jean-Piere Kuilboer, University of Massachusetts, Boston; Daniel Mittleman, DePaul University; Fred Niederman, Saint Louis University; H. Robert Pajkowski, DeVry Institute of Technology, Scarborough, Ontario; June S. Park, University of Iowa; Tom Pettay, DeVry Institute of Technology, Columbus, Ohio; Neil Ramiller, Portland State University; Eliot Rich, University at Albany, State University of New York; Carl Scott, University of Houston; Keng Siau, University of Nebraska–Lincoln; Jonathan Trower, Baylor University; Anna Wachholz, Sheridan College; Randy S. Weinberg, Carnegie Mellon University; Eli J. Weissman, DeVry Institute of Technology, Long Island City, NY; Heinz Roland Weistroffer, Virginia Commonwealth University; Amy Wilson, DeVry Institute of Technology, Decatur, GA; and Vincent C. Yen, Wright State University.

Systems Analysis and Design with UML Version 2.0

AN OBJECT-ORIENTED APPROACH

Second Edition

Alan Dennis
Indiana University

Barbara Haley Wixom
University of Virginia

David Tegarden
Virginia Tech



John Wiley & Sons, Inc.
www.wiley.com/college/dennis

Acquisitions Editor	<i>Beth Lang Golub</i>
Associate Editor	<i>Lorrain Raccuia</i>
Editorial Assistant	<i>Ame Esterline</i>
Marketing Manager	<i>Jillian Rice</i>
Media Editor	<i>Allie K. Morris</i>
Production Manager	<i>Pam Kennedy</i>
Production Editors	<i>Kelly Tavares, Sarah Wolfman-Robichaud</i>
Managing Editor	<i>Kevin Dodds</i>
Illustration Editor	<i>Benjamin Reece</i>
Cover Design	<i>Benjamin Reece</i>
Cover Image	© Digital Vision/Getty Images

This book was set in Minion by Leyh Publishing, LLC and printed and bound by RR Donnelley–Willard. The cover was printed by Phoenix Color Corp.

This book is printed on acid free paper. ☺

Copyright © 2005 John Wiley & Sons, Inc. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc. 222 Rosewood Drive, Danvers, MA 01923, (978)750-8400, fax (978)750-4470. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201)748-6011, fax (201)748-6008, E-Mail: PERMREQ@WILEY.COM.

To order books or for customer service please call 1-800-CALL WILEY (225-5945).

ISBN 0-471-34806-6

WIE ISBN 0-471-65920-7

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

PREFACE

PURPOSE OF THIS BOOK

Systems Analysis and Design (SAD) is an exciting, active field in which analysts continually learn new techniques and approaches to develop systems more effectively and efficiently. However there is a core set of skills that all analysts need to know—no matter what approach or methodology is used. All information systems projects move through the four phases of planning, analysis, design, and implementation; all projects require analysts to gather requirements, model the business needs, and create blueprints for how the system should be built; and all projects require an understanding of organizational behavior concepts like change management and team building. Today, the cost of developing modern software is composed primarily of the cost associated with the developers themselves and not the computers. As such, object-oriented approaches to developing information systems hold much promise in controlling these costs.

Today, the most exciting change to systems analysis and design is the move to object-oriented techniques, which view a system as a collection of self-contained objects that have both data and processes. This change has been accelerated through the creation of the Unified Modeling Language (UML). UML provides a common vocabulary of object-oriented terms and diagramming techniques that is rich enough to model any systems development project from analysis through implementation.

This book captures the dynamic aspects of the field by keeping students focused on doing SAD while presenting the core set of skills that we feel every systems analyst needs to know today and in the future. This book builds on our professional experience as systems analysts and on our experience in teaching SAD in the classroom.

This book will be of particular interest to instructors who have students do a major project as part of their course. Each chapter describes one part of the process, provides clear explanations on how to do it, gives a detailed example, and then has exercises for the students to practice. In this way, students can leave the course with experience that will form a rich foundation for further work as a systems analyst.

OUTSTANDING FEATURES

A Focus on Doing SAD

The goal of this book is to enable students to do SAD—not just read about it, but understand the issues so they can actually analyze and design systems. The book introduces each major technique, explains what it is, explains how to do it, presents an example, and provides opportunities for students to practice before they do it for real in a project. After reading each chapter, the student will be able to perform that step in the system development life cycle (SDLC) process.

Rich Examples of Success and Failure

The book includes a running case about a fictitious company called CD Selections. Each chapter shows how the concepts are applied in situations at CD Selections. Unlike running cases in other books we have tried to focus these examples on planning, managing, and executing the activities described in the chapter, rather than on detailed dialogue between fictitious actors. In this way, the running case serves as a template that students can apply to their own work. Each chapter also includes numerous Concepts in Action boxes that describe how real companies succeeded—and failed—in performing the activities in the chapter. Many of these examples are drawn from our own experiences as systems analysts.

Real World Focus

The skills that students learn in a systems analysis and design course should mirror the work that they ultimately will do in real organizations. We have tried to make this book as “real” as possible by building extensively on our experience as professional systems analysts for organizations such as Arthur Andersen, IBM, the U.S. Department of Defense, and the Australian Army. We have also worked with a diverse industry advisory board of IS professionals and consultants in developing the book and have incorporated their stories, feedback, and advice throughout. Many students who use this book will eventually use the skills on the job in a business environment, and we believe they will have a competitive edge in understanding what successful practitioners feel is relevant in the real world.

Project Approach

We have presented the topics in this book in the SDLC order in which an analyst encounters them in a typical project. Although the presentation is necessarily linear (because students have to learn concepts in the way in which they build on each other), we emphasize the iterative, complex nature of SAD as the book unfolds. The presentation of the material should align well with courses that encourage students to work on projects because it presents topics as students need to apply them.

WHAT'S NEW IN THIS EDITION

This new edition has four major improvements. First, we converted the entire text from UML 1.3 to UML 2.0. Due to the size and complexity of UML 2.0, we factored the object-oriented material out of Chapter 1 and created a new chapter (Chapter 2) that includes an introduction to object orientation, UML 2.0, the Unified Process, and which overviews a minimalist approach to Object-Oriented Systems Analysis and Design with UML 2.0. Some of this material originally was included in Chapter 9. However, for continuity and flexibility purposes, we moved this material to Chapter 2. This new chapter is organized in such a way to allow the instructor the maximum amount of flexibility to choose what material they want to emphasize. For example, if the students have already had an object-oriented programming course using Java, the basic characteristics of object-oriented systems section could either be assigned for reading or review only. Or, if the instructor would prefer to minimize the amount of UML to be introduced to the student at this point in the course, the UML 2.0 section could be optional. The details of the relevant UML 2.0 are covered in the book where the diagrams are applicable, e.g., class diagrams are covered in detail

in Chapter 7: Structural Models. In this section of Chapter 2, we only introduce the purpose of the fourteen diagrams that are now included in the UML.

Second, not only have we updated the text to cover UML 2.0, we also have expanded the coverage of UML. In this edition, we have added activity diagrams to support business process modeling (see Chapter 6) and deployment diagrams to provide for modeling the physical architecture of the system (see Chapter 13).

Third, we combined some chapters together to improve the flow of material through the book. For example, in the first edition we had two chapters devoted to user interface design. In this edition we have combined the material into a single chapter.

Fourth, we reordered the material dealing with object-oriented systems design contained in Part Two. The new order is class and method design (Chapter 10), object storage design (Chapter 11), user interface design (Chapter 12), and finally, the design of the physical architecture of the system (Chapter 13). The new order provides a more logical organization of this material.

Of course, one feature that we were sure to keep was to keep the footnotes in the chapters. Even though at first glance footnotes do not seem to add a lot to a textbook, they provide a means for the student and/or instructor to delve deeper into any topic included in this text. With the expanded material, we have added quite a few footnotes throughout the text.

ORGANIZATION OF THIS BOOK

This book is organized by the phases of the Systems Development Life Cycle (SDLC). Each chapter has been written to teach students specific tasks that analysts need to accomplish over the course of a project, and the deliverables that will be produced from the tasks. As students complete the book, tasks will be “checked off” and deliverables will be completed. Along the way, students will be reminded of their progress using roadmaps that indicate where their current task fits into the larger context of SAD.

Chapter 1 introduces the SDLC and describes the roles and skills needed for a project team. Chapter 2 introduces the basic characteristics of object-oriented systems, UML 2.0, Object-Oriented Systems Analysis, and the Unified Process. It also overviews a minimalist approach to Object-Oriented Systems Analysis and Design with UML 2.0.

Part One contains Chapters 3 and 4 which describe the Planning Phase. Chapter 3 presents Project Initiation, with a focus on the System Request, Feasibility Analysis, and Project Selection. In Chapter 4, students learn about Project Management, with emphasis on the Workplan, Staffing Plan, Project Charter, and Risk Assessment that are used to help manage and control the project.

Part Two presents techniques needed during the Analysis Phase. In Chapter 5, students are introduced to a variety of requirements gathering techniques that are used to create an Analysis Plan after learning an assortment of analysis techniques to help with Business Automation, Business Improvement, and Business Process Reengineering. Using the identified requirements, Chapter 6 focuses on constructing Functional Models, Chapter 7 addresses producing Structural Models, and Chapter 8 tackles creating Behavioral Models.

Part Three addresses the Design Phase. In Chapter 9, students learn how to evolve the analysis models into design models via the use of factoring, partitions, and layers. The students also learn to create an Alternative Matrix that can be used to compare custom, packaged, and outsourcing alternatives. Chapter 10 concentrates on designing the individual

classes and their respective methods through the use of contracts and method specifications. Chapter 11 presents the issues involved in designing persistence for objects. These issues include the different storage formats that can be used for object persistence, how to map an object-oriented design into the chosen storage format, and how to design a set of data access and manipulation classes that act as a translator between the classes in the application and the object persistence. Chapter 12 presents the design of the human computer interaction layer where students learn how to design user interfaces using Use Scenarios, Windows Navigation Diagrams, Real Use Cases, Interface Standards, and User Interface Templates. Chapter 13 focuses on the physical architecture design, which includes Deployment Diagrams and Hardware/Software Specification.

Part Four provides material that is related to the Implementation Phase. Chapter 14 focuses on system construction where students learn how to build, test, and document the system. Installation and operations are covered in Chapter 15 where students learn about the Conversion Plan, Change Management Plan, Support Plan, and Project Assessment.

SUPPLEMENTS <http://www.wiley.com/college/dennis>

Instructor's Resources Web Site

- PowerPoint slides, that instructors can tailor to their classroom needs and that students can use to guide their reading and studying activities
- Test Bank, that includes a variety of questions ranging from multiple choice to essay style questions. A computerized version of the Test Bank will also be available.

Online Instructor's Manual

The Instructor's Manual provides resources to support the instructor both inside and out of the classroom:

- Short experiential exercises that instructors can use to help students experience and understand key topics in each chapter.
- Short stories have been provided by people working in both corporate and consulting environments for instructors to insert into lectures to make concepts more colorful and real
- Additional mini-cases for every chapter allow students to perform some of the key concepts that were learned in the chapter.
- Solutions to end of chapter questions and exercises are provided.

Student Web Site

- Relevant Web links, including career resources Web site.
- Web Quizzes help students prepare for class tests.

Cases in Systems Analysis and Design

A separate Case Book on CD-ROM, provides a set of more than a dozen cases that can be used to supplement the book and provide exercises for students to practice with. The cases are primarily drawn from the U.S. and Canada, but also include a number of international cases. We are always looking for new cases, so if you have a case that might be appropriate please contact us directly (or your local Wiley sales representative).

Software Tools

Five Software Tools can be purchased with the text in special packages:

1. Oracle's 9i Database Suite, which includes Oracle 9i Personal Edition, Oracle 9i Enterprise Edition, and Oracle 9i Standard Edition.
2. Oracle's Developer Suite, which includes Oracle Developer and Oracle Designer. This software is available under a "Development Sublicense" for personal development purposes only, and has no time restrictions or limitations.
3. Visible Systems Corporation's Visible Analyst Student Edition.
4. Microsoft's Visio
5. Microsoft's Project

Contact your local Wiley sales representative for details, including pricing and ordering information.

ACKNOWLEDGEMENTS

Thanks to Roberta Roth, University of Northern Iowa and Suresh Chalasani of University of Wisconsin Parkside, for their work on the supplements.

We would like to thank the following reviewers for their helpful and insightful comments on the second edition: Murugan Anandarajon, Drexel University; Ron Anson, Boise State University; Noushin Ashrafi, University of Massachusetts Boston; Dirk Baldwin, University of Wisconsin; Robert Barker, University of Louisville; Terry Fox, Baylor University; Donald Golden, Cleveland State University; Cleotilde Gonzalez, Carnegie Mellon University; Scott James, Saginaw Valley State University; Rajiv Kishore, State University of New York–Buffalo; Ravindra Krovi, University of Akron; Fernando Maymi, United States Military Academy at West Point; Fred Niederman, Saint Louis University; Graham Peace, West Virginia University; J. Drew Procaccino, Rider University; Marcus Rothenberger, University of Wisconsin–Milwaukee; June Verner, Drexel University; Heinz Roland Weistroffer, Virginia Commonwealth University; and Amy Woszcynski, Kennesaw State University.

We also thank the following reviewers from the first edition: Evans Adams, Fort Lewis College; Noushin Ashrafi, University of Massachusetts, Boston; Dirk Baldwin, University of Wisconsin-Parkside; Qing Cao, University of Missouri–Kansas City; Ahmad Ghafarian, North Georgia College & State University; Daniel V. Goulet, University of Wisconsin–Stevens Point; Harvey Hayashi, Loyalist College of Applied Arts and Technology; Jean-Piere Kuilboer, University of Massachusetts, Boston; Daniel Mittleman, DePaul University; Fred Niederman, Saint Louis University; H. Robert Pajkowski, DeVry Institute of Technology, Scarborough, Ontario; June S. Park, University of Iowa; Tom Pettay, DeVry Institute of Technology, Columbus, Ohio; Neil Ramiller, Portland State University; Eliot Rich, University at Albany, State University of New York; Carl Scott, University of Houston; Keng Siau, University of Nebraska–Lincoln; Jonathan Trower, Baylor University; Anna Wachholz, Sheridan College; Randy S. Weinberg, Carnegie Mellon University; Eli J. Weissman, DeVry Institute of Technology, Long Island City, NY; Heinz Roland Weistroffer, Virginia Commonwealth University; Amy Wilson, DeVry Institute of Technology, Decatur, GA; and Vincent C. Yen, Wright State University.

CHAPTER 1

INTRODUCTION TO SYSTEMS ANALYSIS AND DESIGN

This chapter introduces the systems development life cycle (SDLC), the fundamental four-phase model (planning, analysis, design, and implementation) that is common to all information system development projects. It then describes the evolution of system development methodologies. Finally, the chapter closes with a discussion of the roles and skills necessary within the project team.

OBJECTIVES

- Understand the fundamental systems development life cycle and its four phases.
- Understand the evolution of systems development methodologies.
- Be familiar with the different roles on the project team.

CHAPTER OUTLINE

Introduction	Selecting the Appropriate Development Methodology
The Systems Development Life Cycle	Project Team Roles and Skills
Planning	Business Analyst
Analysis	Systems Analyst
Design	Infrastructure Analyst
Implementation	Change Management Analyst
Systems Development Methodologies	Project Manager
Structured Design	Summary
Rapid Application Development (RAD)	
Agile Development	

INTRODUCTION

The *systems development life cycle (SDLC)* is the process of understanding how an information system (IS) can support business needs, designing the system, building it, and delivering it to users. If you have taken a programming class or have programmed on your own, this probably sounds pretty simple. Unfortunately, this is not the case. A 1996 survey by the Standish Group found that 42 percent of all corporate IS projects were abandoned before completion. A similar study done in 1996 by the General Accounting Office found 53 percent of all U.S. government IS projects were abandoned. Unfortunately, many of the

2 Chapter 1 Introduction to Systems Analysis and Design

systems that aren't abandoned are delivered to the users significantly late, cost far more than planned, and have fewer features than originally planned.

Most of us would like to think that these problems only occur to "other" people or "other" organizations, but they happen in most companies. Even Microsoft has a history of failures and overdue projects (e.g., Windows 1.0, Windows 95).¹

Although we would like to promote this book as a "silver bullet" that will keep you from IS failures, we readily admit that a silver bullet that guarantees IS development success simply does not exist.² Instead, this book will provide you with several fundamental concepts and many practical techniques that you can use to improve the probability of success.

The key person in the SDLC is the systems analyst who analyzes the business situation, identifies opportunities for improvements, and designs an information system to implement them. Being a systems analyst is one of the most interesting, exciting, and challenging jobs around. As a systems analyst, you will work with a variety of people and learn how they conduct business. Specifically, you will work with a team of systems analysts, programmers, and others on a common mission. You will feel the satisfaction of seeing systems that you designed and developed make a significant business impact, while knowing that you contributed your unique skills to make that happen.

It is important to remember that the primary objective of the systems analyst is not to create a wonderful system. The primary goal is to create value for the organization, which for most companies means increasing profits (government agencies and not-for-profit organizations measure value differently). Many failed systems were abandoned because the analysts tried to build a wonderful system without clearly understanding how the system would fit with the organization's goals, current business processes, and other information systems to provide value. An investment in an information system is like any other investment, such as a new machine tool. The goal is not to acquire the tool, because the tool is simply a means to an end; the goal is to enable the organization to perform work better so it can earn greater profits or serve its constituents more effectively.

This book will introduce you to the fundamental skills you need to be a systems analyst. This is a pragmatic book that discusses best practices in systems development; it does not present a general survey of systems development that exposes you to everything about the topic. By definition, systems analysts *do things* and challenge the current way that organizations work. To get the most out of this book, you will need to actively apply the ideas and concepts in the examples, and those in the "Your Turn" exercises that are presented throughout, to your own systems development project. This book will guide you through all the steps for delivering a successful information system. Also, we will illustrate how one organization (which we call CD Selections) applies the steps in one project (developing a Web-based CD sales system). By the time you finish the book, you won't be an expert analyst, but you will be ready to start building systems for real.

In this chapter, we first introduce the basic SDLC that IS projects follow. This life cycle is common to all projects, although the focus and approach to each phase of the life cycle may differ. In the next section, we describe three fundamentally different types of system development methodologies: structured design, rapid application development, and agile development. Finally, we discuss one of the most challenging aspects of systems

¹ For more information on the problem, see Capers Jones, *Patterns of Software System Failure and Success* (London: International Thompson Computer Press, 1996); Capers Jones, *Assessment and Control of Software Project Risks* (Englewood Cliffs, NJ: Yourdon Press, 1994); Julia King, "IS reins in runaway projects," *Computerworld* (February 24, 1997).

² The idea of using the silver bullet metaphor was first described in a paper by Frederick Brooks, see Frederick P. Brooks, Jr., "No Silver Bullet – Essence and Accident in Software Engineering," *Information Processing 1986, the Proceedings of the IFIP Tenth World Computing Conference*, edited by H.-J. Kugler (1986): 1069-76.

development, the depth and breadth of skills that are required of systems analysts. Today, most organizations use project teams that contain members with unique, but complementary skills. This chapter closes with a discussion of the key roles played by members of the systems development team.

THE SYSTEMS DEVELOPMENT LIFE CYCLE

In many ways, building an information system is similar to building a house. First, the house (or the information system) starts with a basic idea. Second, this idea is transformed into a simple drawing that is shown to the customer and refined (often through several drawings, each improving on the other) until the customer agrees that the picture depicts what he or she wants. Third, a set of blueprints is designed that presents much more detailed information about the house (e.g., the type of water faucets, where the telephone jacks will be placed). Finally, the house is built following the blueprints—and often with some changes directed by the customer as the house is erected.

The SDLC has a similar set of four fundamental *phases*: planning, analysis, design, and implementation. Different projects may emphasize different parts of the SDLC or approach the SDLC phases in different ways, but all projects have elements of these four phases. Each *phase* is itself composed of a series of *steps*, which rely upon *techniques* that produce *deliverables* (specific documents and files that provide understanding about the project).

For example, when you apply for admission to a university, there are several phases that all students go through: information gathering, applying, and accepting. Each of these phases has steps—information gathering includes steps like searching for schools, requesting information, and reading brochures. Students then use techniques (e.g., Internet searching) that can be applied to steps (e.g., requesting information) to create deliverables (e.g., evaluations of different aspects of universities).

In many projects, the SDLC phases and steps proceed in a logical path from start to finish. In other projects, the project teams move through the steps consecutively, incrementally, iteratively, or in other patterns. In this section, we describe the phases, steps, and some of the techniques that are used to accomplish the steps at a very high level. We should emphasize that not all organizations follow the SDLC in exactly the same way. As we shall shortly see, there are many variations on the overall SDLC.

CONCEPTS

1-A An Expensive False Start

IN ACTION

A real-estate group in the federal government cosponsored a data warehouse with the IT department. A formal proposal was written by IT in which costs were estimated at \$800,000, the project duration was estimated to be eight months, and the responsibility for funding was defined as the business unit's. The IT department proceeded with the project before it even knew if the project had been accepted.

The project actually lasted two years because requirements gathering took nine months instead of one and a half, the planned user base grew from 200 to 2,500, and the approval process to buy technology for the project

took a year. Three weeks prior to technical delivery, the IT director canceled the project. This failed endeavor cost the organization and taxpayers \$2.5 million.

Source: Hugh J. Watson et al., "Data Warehousing Failure: Case Studies and Findings," *The Journal of Data Warehousing* 4 (1) (1999): 44–54.

Question:

1. Why did this system fail? Why would a company spend money and time on a project and then cancel it? What could have been done to prevent this?

For now, there are two important points to understand about the SDLC. First, you should get a general sense of the phases and steps that IS projects move through and some of the techniques that produce certain deliverables. Second, it is important to understand that the SDLC is a process of *gradual refinement*. The deliverables produced in the analysis phase provide a general idea of the shape of the new system. These deliverables are used as input to the design phase, which then refines them to produce a set of deliverables that describes in much more detailed terms exactly how the system will be built. These deliverables, in turn, are used in the implementation phase to produce the actual system. Each phase refines and elaborates on the work done previously.

Planning

The *planning phase* is the fundamental process of understanding *why* an information system should be built and determining how the project team will go about building it. It has two steps:

1. During *project initiation*, the system's business value to the organization is identified—how will it lower costs or increase revenues? Most ideas for new systems come from outside the IS area (from the marketing department, accounting department, etc.) in the form of a system request. A *system request* presents a brief summary of a business need, and it explains how a system that supports the need will create business value. The IS department works together with the person or department that generated the request (called the *project sponsor*) to conduct a feasibility analysis. The *feasibility analysis* examines key aspects of the proposed project:
 - The technical feasibility (Can we build it?)
 - The economic feasibility (Will it provide business value?)
 - The organizational feasibility (If we build it, will it be used?)

The system request and feasibility analysis are presented to an information systems *approval committee* (sometimes called a *steering committee*), which decides whether the project should be undertaken.

2. Once the project is approved, it enters—*project management*. During project management, the *project manager* creates a *workplan*, staffs the project, and puts techniques in place to help the project team control and direct the project through the entire SDLC. The deliverable for project management is a *project plan* that describes how the project team will go about developing the system.

Analysis

The *analysis phase* answers the questions of *who* will use the system, *what* the system will do, and *where* and *when* it will be used. During this phase, the project team investigates any current system(s), identifies improvement opportunities, and develops a concept for the new system. This phase has three steps:

1. An *analysis strategy* is developed to guide the project team's efforts. Such a strategy usually includes an analysis of the current system (called the *as-is system*) and its problems, and then ways to design a new system (called the *to-be system*).
2. The next step is *requirements gathering* (e.g., through interviews or questionnaires). The analysis of this information—in conjunction with input from project sponsor and many other people—leads to the development of a concept for a new system. The system concept is then used as a basis to develop a set of business

analysis models that describes how the business will operate if the new system were developed. The set of models typically includes models that represent the data and processes necessary to support the underlying business process.

3. The analyses, system concept, and models are combined into a document called the *system proposal*, which is presented to the project sponsor and other key decision makers (e.g., members of the approval committee) that decide whether the project should continue to move forward.

The system proposal is the initial deliverable that describes what business requirements the new system should meet. Because it is really the first step in the design of the new system, some experts argue that it is inappropriate to use the term *analysis* as the name for this phase; some argue a better name would be *analysis and initial design*. Most organizations continue use to the name *analysis* for this phase, however, so we will use it in this book as well. Just keep in mind that the deliverable from the analysis phase is both an analysis and a high-level initial design for the new system.

Design

The *design phase* decides *how* the system will operate, in terms of the hardware, software, and network infrastructure; the user interface, forms and reports; and the specific programs, databases, and files that will be needed. Although most of the strategic decisions about the system were made in the development of the system concept during the analysis phase, the steps in the design phase determine exactly how the system will operate. The design phase has four steps:

1. The *design strategy* must be developed. This clarifies whether the system will be developed by the company's own programmers, whether it will be outsourced to another firm (usually a consulting firm), or whether the company will buy an existing software package.
2. This leads to the development of the basic *architecture design* for the system that describes the hardware, software, and network infrastructure that will be used. In most cases, the system will add or change the infrastructure that already exists in the organization. The *interface design* specifies how the users will move through the system (e.g., navigation methods such as menus and on-screen buttons) and the forms and reports that the system will use.
3. The *database and file specifications* are developed. These define exactly what data will be stored and where they will be stored.
4. The analyst team develops the *program design*, which defines the programs that need to be written and exactly what each program will do.

This collection of deliverables (architecture design, interface design, database and file specifications, and program design) is the *system specification* that is handed to the programming team for implementation. At the end of the design phase, the feasibility analysis and project plan are reexamined and revised, and another decision is made by the project sponsor and approval committee about whether to terminate the project or continue.

Implementation

The final phase in the SDLC is the *implementation phase*, during which the system is actually built (or purchased, in the case of a packaged software design). This is the phase that usually gets the most attention, because for most systems it is longest and most expensive single part of the development process. This phase has three steps:

1. System *construction* is the first step. The system is built and tested to ensure it performs as designed. Since the cost of bugs can be immense, testing is one of the most critical steps in implementation. Most organizations spend more time and attention on testing than on writing the programs in the first place.
2. The system is installed. *Installation* is the process by which the old system is turned off and the new one is turned on. It may include a direct cutover approach (in which the new system immediately replaces the old system), a parallel conversion approach (in which both the old and new systems are operated for a month or two until it is clear that there are no bugs in the new system), or a phased conversion strategy (in which the new system is installed in one part of the organization as an initial trial and then gradually installed in others). One of the most important aspects of conversion is the development of a *training plan* to teach users how to use the new system and help manage the changes caused by the new system.
3. The analyst team establishes a *support plan* for the system. This plan usually includes a formal or informal post-implementation review, as well as a systematic way for identifying major and minor changes needed for the system.

SYSTEMS DEVELOPMENT METHODOLOGIES

A *methodology* is a formalized approach to implementing the SDLC (i.e., it is a list of steps and deliverables). There are many different systems development methodologies, and each one is unique based on the order and focus it places on each SDLC phase. Some methodologies are formal standards used by government agencies, while others have been developed by consulting firms to sell to clients. Many organizations have internal methodologies that have been honed over the years, and they explain exactly how each phase of the SDLC is to be performed in that company.

There are many ways to categorize methodologies. One way is by looking at whether they focus on business processes or the data that support the business. Methodologies are *process-centered* if they emphasize process models as the core of the system concept. In Figure 1-1, for example, process-centered methodologies would focus first on defining the processes (e.g., assemble sandwich ingredients). Methodologies are *data-centered* if they emphasize data models as the core of the system concept. In Figure 1-1, for example, data-centered methodologies would focus first on defining the contents of the storage areas (e.g., refrigerator) and how the contents were organized.³ By contrast, *object-oriented methodologies* attempt to balance the focus between process and data by incorporating both into one model. In Figure 1-1, these methodologies would focus first on defining the major elements of the system (e.g., sandwiches, lunches) and look at the processes and data that were involved with each element.

Another important factor in categorizing methodologies is the sequencing of the SDLC phases and the amount of time and effort devoted to each.⁴ In the early days of

³ The classic modern process-centered methodology is that by Edward Yourdon, *Modern Structured Analysis* (Englewood Cliffs, NJ: Yourdon Press, 1989). An example of a data-centered methodology is information engineering by James Martin, *Information Engineering*, vols. 1-3 (Englewood Cliffs, NJ: Prentice Hall, 1989). A widely accepted standardized non-object-oriented methodology that balances processes and data is IDEF; see FIPS 183, *Integration Definition for Function Modeling*, Federal Information Processing Standards Publications, U.S. Department of Commerce, 1993.

⁴ A good reference for comparing systems development methodologies is Steve McConnell, *Rapid Development* (Redmond, WA: Microsoft Press, 1996).

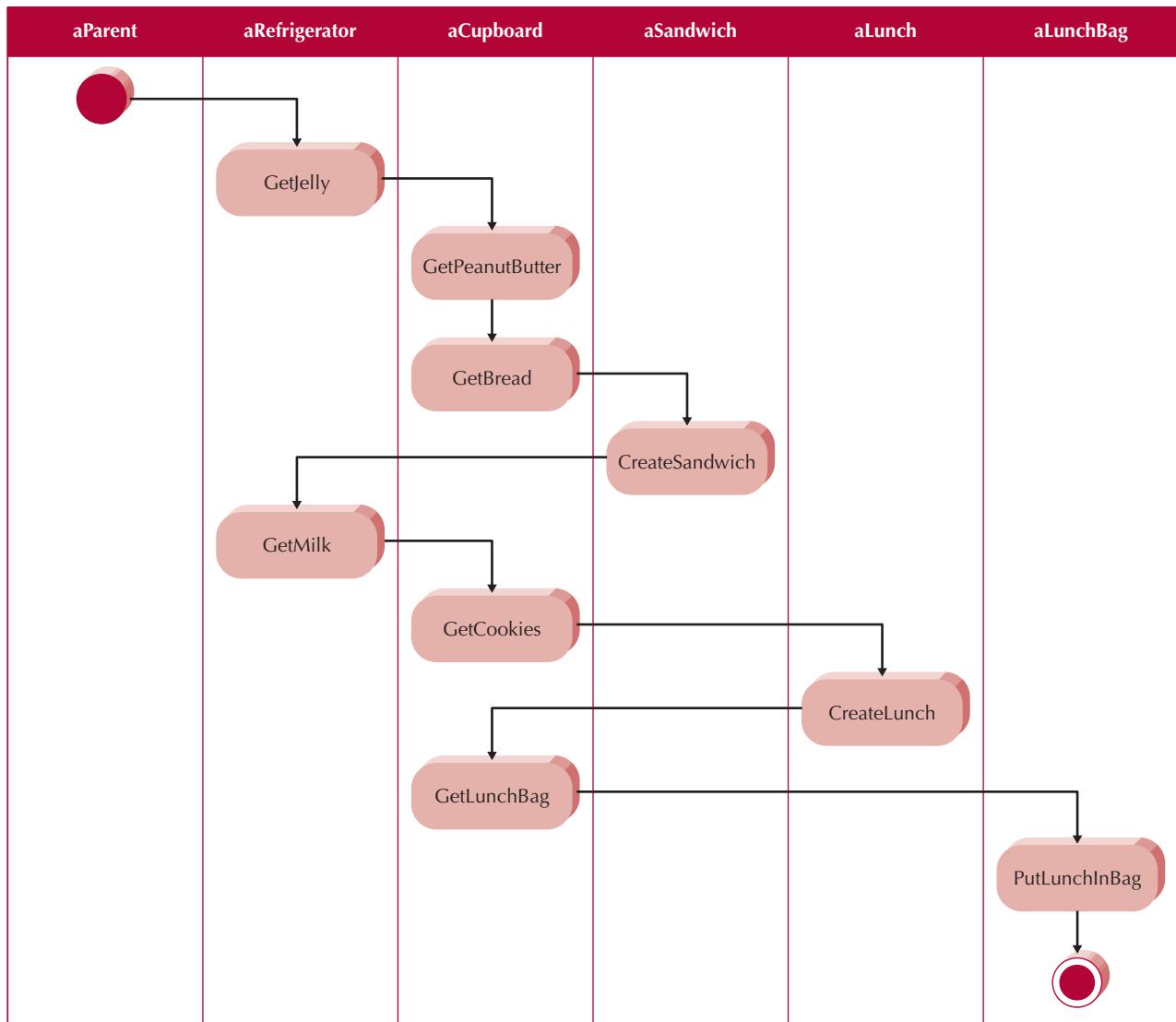


FIGURE 1-1 A Simple Behavioral Model for Making Lunch

computing, the need for formal and well-planned life cycle methodologies was not well understood. Programmers tended to move directly from a very simple planning phase right into the construction step of the implementation phase. In other words, they moved directly from a very fuzzy, not well-thought-out system request into writing code.

This is the same approach that you sometimes use when writing programs for a programming class. It can work for small programs that require only one programmer, but if the requirements are complex or unclear, you may miss important aspects of the problem and have to start all over again, throwing away part of the program (and the time and effort spent writing it). This approach also makes teamwork difficult because members have little idea about what needs to be accomplished and how to work together to produce a final product.

Structured Design

The first category of systems development methodologies is called *structured design*. These methodologies became dominant in the 1980s, replacing the previous, ad hoc, and undisciplined approaches. Structured design methodologies adopt a formal step-by-step approach to the SDLC that moves logically from one phase to the next. Numerous process-centered and data-centered methodologies follow the basic approach of the two structured design categories outlined next.

Waterfall Development The original structured design methodology (that is still used today) is *waterfall development*. With waterfall development-based methodologies, the analysts and users proceed in sequence from one phase to the next (see Figure 1-2). The key deliverables for each phase are typically very long (often hundreds of pages in length) and are presented to the project sponsor for approval as the project moves from phase to phase. Once the sponsor approves the work that was conducted for a phase, the phase ends and the next one begins. This methodology is referred to as waterfall development because it moves forward from phase to phase in the same manner as a waterfall. While it is possible to go backward in the SDLC (e.g., from design back to analysis), it is extremely difficult (imagine yourself as a salmon trying to swim “upstream” against a waterfall, as shown in Figure 1-2).

Structured design also introduced the use of formal modeling or diagramming techniques to describe the basic business processes and the data that support them. Traditional structured design uses one set of diagrams to represent the processes and a separate set of diagrams to represent data. Because two sets of diagrams are used, the systems analyst must decide which set to develop first and use as the core of the system—process-model diagrams or data-model diagrams. There is much debate over which should come first, the processes or the data, because both are important to the system. As a result, several different structured design methodologies have evolved that follow the basic steps of the waterfall model, but use different modeling approaches at different times. Those that attempt to emphasize process-model diagrams as the core of the system are process-centered, while those that emphasize data-model diagrams as the core of the system concept are data-centered.

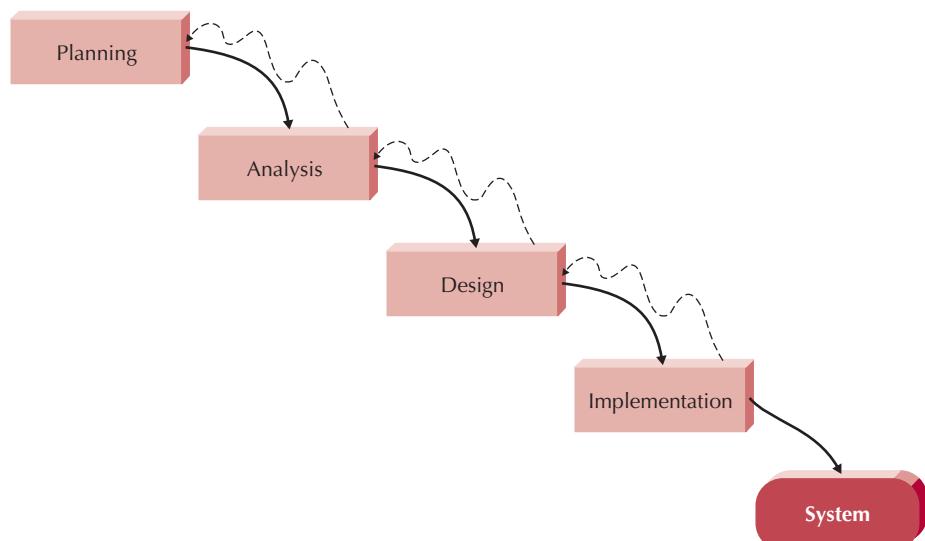


FIGURE 1-2
A Waterfall
Development-based
Methodology

The two key advantages of the structured design waterfall approach are that it identifies system requirements long before programming begins and it minimizes changes to the requirements as the project proceeds. The two key disadvantages are that the design must be completely specified before programming begins and that a long time elapses between the completion of the system proposal in the analysis phase and the delivery of the system (usually many months or years). The lengthy deliverables are often a poor communication mechanism; the result is that important requirements can be overlooked in the voluminous documentation. Users rarely are prepared for their introduction to the new system, which occurs long after the initial idea for the system was introduced. If the project team misses important requirements, expensive post-implementation programming may be needed (imagine yourself trying to design a car on paper; how likely would you be to remember interior lights that come on when the doors open, or to specify the right number of valves on the engine?).

A system also may require significant rework because the business environment has changed from the time that the analysis phase occurred. When changes do occur, it means going back to the initial phases and following the change through each of the subsequent phases in turn.

Parallel Development The *parallel development* methodology attempts to address the problem of long delays between the analysis phase and the delivery of the system. Instead of doing design and implementation in sequence, it performs a general design for the whole system and then divides the project into a series of distinct subprojects that can be designed and implemented in parallel. Once all subprojects are complete, there is a final integration of the separate pieces, and the system is delivered (see Figure 1-3).

The primary advantage of this methodology is that it can reduce the schedule time to deliver a system; thus, there is less chance of changes in the business environment causing rework. However, the approach still suffers from problems caused by paper documents. It also adds a new problem: Sometimes the subprojects are not completely independent; design decisions made in one subproject may affect another, and the end of the project may require significant integration efforts.

Rapid Application Development (RAD)

A second category of methodologies includes *rapid application development (RAD)*—based methodologies. These are a newer class of systems development methodologies that emerged in the 1990s. RAD-based methodologies attempt to address both weaknesses of structured design methodologies by adjusting the SDLC phases to get some part of the system developed quickly and into the hands of the users. In this way, the users can better understand the system and suggest revisions that bring the system closer to what is needed.⁵

Most RAD-based methodologies recommend that analysts use special techniques and computer tools to speed up the analysis, design, and implementation phases, such as CASE tools (see Chapter 4), joint application design (JAD) sessions (see Chapter 5), fourth generation/visual programming languages that simplify and speed-up programming (e.g., Visual Basic), and code generators that automatically produce programs from design specifications. It is the combination of the changed SDLC phases and the use of these tools and techniques that improve the speed and quality of systems development. However, there is one

⁵ One of the best RAD books is that by Steve McConnell, *Rapid Development* (Redmond, WA: Microsoft Press, 1996).

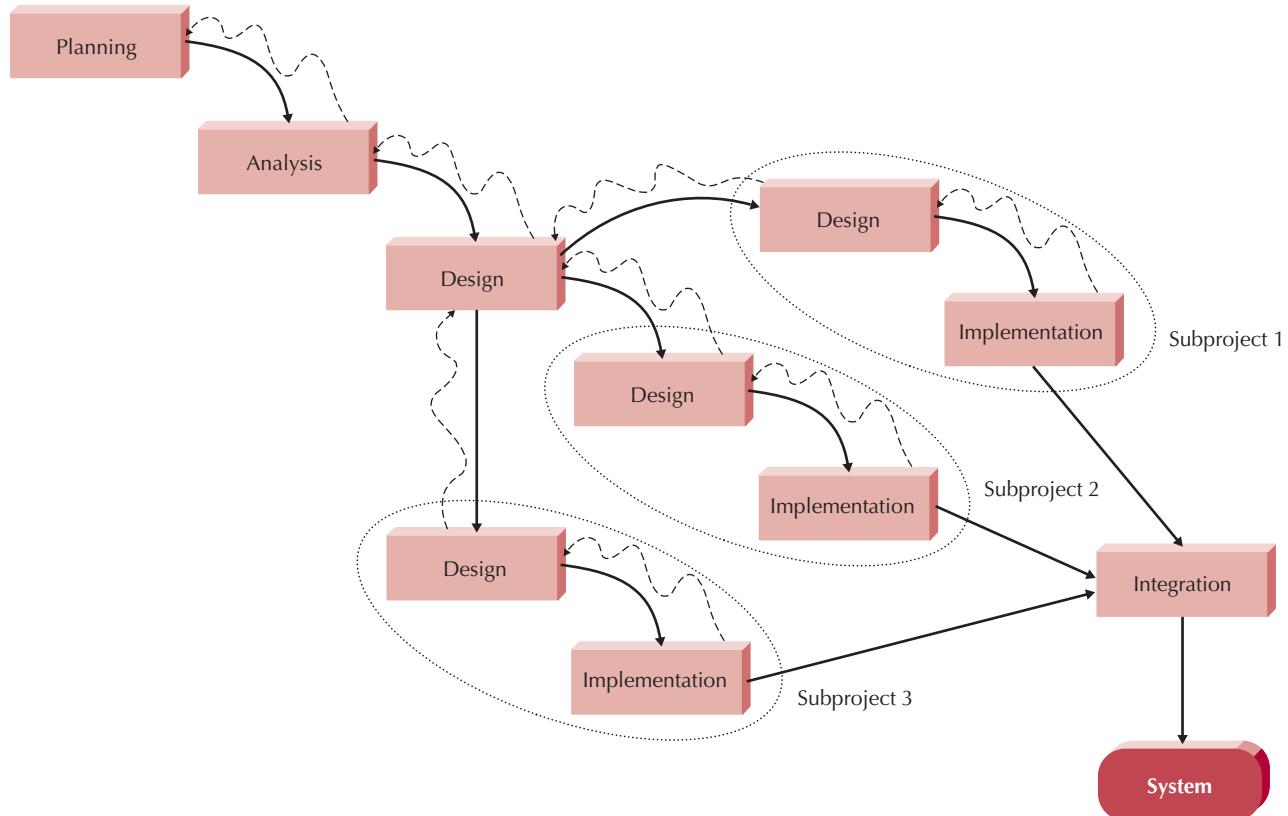


FIGURE 1-3 A Parallel Development-based Methodology

possible subtle problem with RAD-based methodologies: user expectation management. Due to the use of the tools and techniques that can improve the speed and quality of systems development, user expectations of what is possible may dramatically change. As the user better understands the information technology, the systems requirements tend to expand. This was less of a problem when using methodologies that spent a lot of time thoroughly documenting requirements. There are process-centered, data-centered and object-oriented methodologies that follow the basic approaches of the three RAD categories described below.

Phased Development A *phased development*-based methodology breaks the overall system into a series of *versions* that are developed sequentially. The analysis phase identifies the overall system concept, and the project team, users, and system sponsor then categorize the requirements into a series of versions. The most important and fundamental requirements are bundled into the first version of the system. The analysis phase then leads into design and implementation, but only with the set of requirements identified for version 1 (see Figure 1-4).

Once version 1 is implemented, work begins on version 2. Additional analysis is performed based on the previously identified requirements and combined with new ideas and issues that arose from the users' experience with version 1. Version 2 then is designed and implemented, and work immediately begins on the next version. This process continues until the system is complete or is no longer in use.

Phased development-based methodologies have the advantage of quickly getting a useful system into the hands of the users. While the system does not perform all the functions

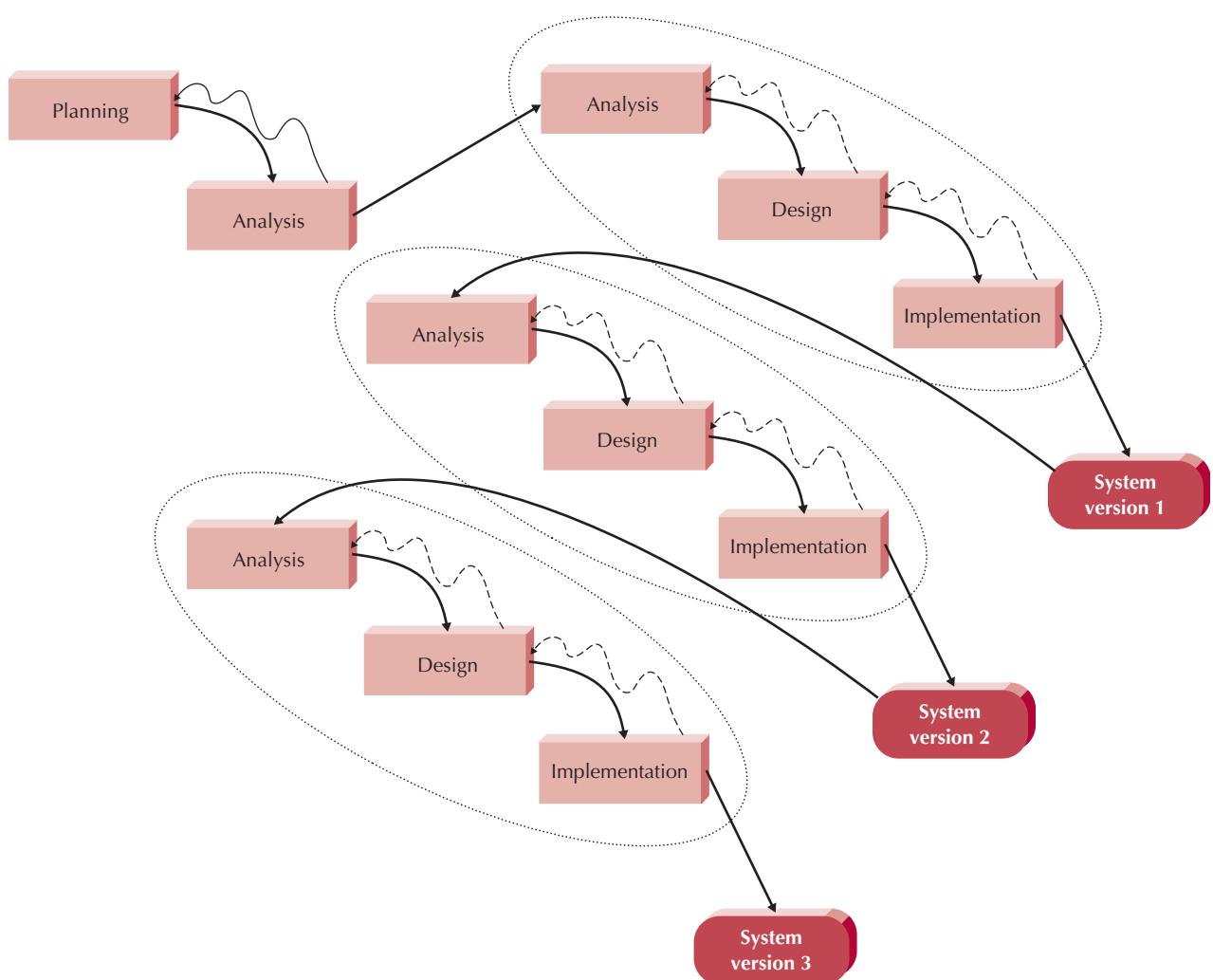


FIGURE 1-4 A Phased Development-based Methodology

the users need at first, it does begin to provide business value sooner than if the system were delivered after completion, as is the case with the waterfall or parallel methodology. Likewise, because users begin to work with the system sooner, they are more likely to identify important additional requirements sooner than with structured design situations.

The major drawback to phased development is that users begin to work with systems that are intentionally incomplete. It is critical to identify the most important and useful features and include them in the first version, while managing users' expectations along the way.

Prototyping A *prototyping*-based methodology performs the analysis, design, and implementation phases concurrently, and all three phases are performed repeatedly in a cycle until the system is completed. With these methodologies, the basics of analysis and design are performed, and work immediately begins on a *system prototype*, a “quick-and-dirty” program that provides a minimal amount of features. The first prototype is usually the first part of the system that the user will use. This is shown to the users and

project sponsor who provide comments, which are used to re-analyze, re-design, and re-implement a second prototype that provides a few more features. This process continues in a cycle until the analysts, users, and sponsor agree that the prototype provides enough functionality to be installed and used in the organization. After the prototype (now called the *system*) is installed, refinement occurs until it is accepted as the new system (see Figure 1-5).

The key advantage of a prototyping-based methodology is that it *very* quickly provides a system for the users to interact with, even if it is not ready for widespread organizational use at first. Prototyping reassures the users that the project team is working on the system (there are no long delays in which the users see little progress), and prototyping helps to more quickly refine real requirements. Rather than attempting to understand a system specification on paper, the users can interact with the prototype to better understand what it can and cannot do.

The major problem with prototyping is that its fast-paced system releases challenge attempts to conduct careful, methodical analysis. Often the prototype undergoes such significant changes that many initial design decisions become poor ones. This can cause problems in the development of complex systems because fundamental issues and problems are not recognized until well into the development process. Imagine building a car and discovering late in the prototyping process that you have to take the whole engine out to change the oil (because no one thought about the need to change the oil until after it had been driven 10,000 miles).

Throwaway Prototyping Throwaway prototyping-based methodologies are similar to prototyping-based methodologies in that they include the development of prototypes; however throwaway prototypes are done at a different point in the SDLC. These prototypes are used for a very different purpose than ones previously discussed, and they have a very different appearance (see Figure 1-6).

The throwaway prototyping-based methodologies have a relatively thorough analysis phase that is used to gather information and to develop ideas for the system concept. However, many of the features suggested by the users may be not well understood, and there may be challenging technical issues to be solved. Each of these issues is examined by analyzing, designing, and building a *design prototype*. A design prototype is not a working system; it is a product that represents a part of the system that needs additional refinement, and it only contains enough detail to enable users to understand the issues under consideration. For example, suppose users are not completely clear on how an order entry system should work. The analyst team might build a series of HTML pages viewed using

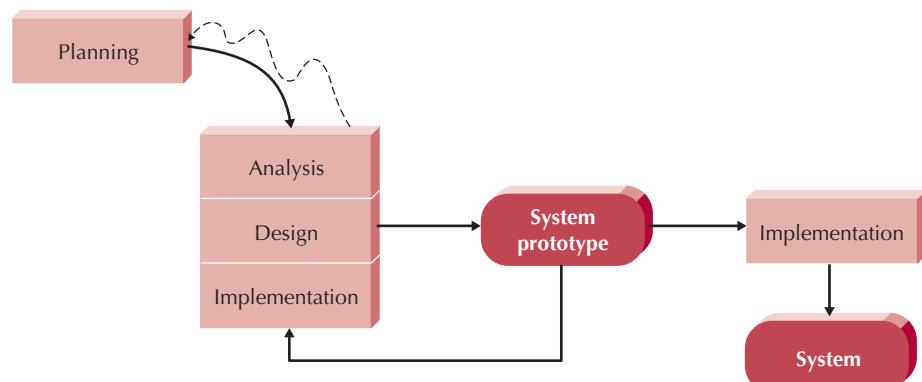


FIGURE 1-5
A Prototyping-based
Methodology

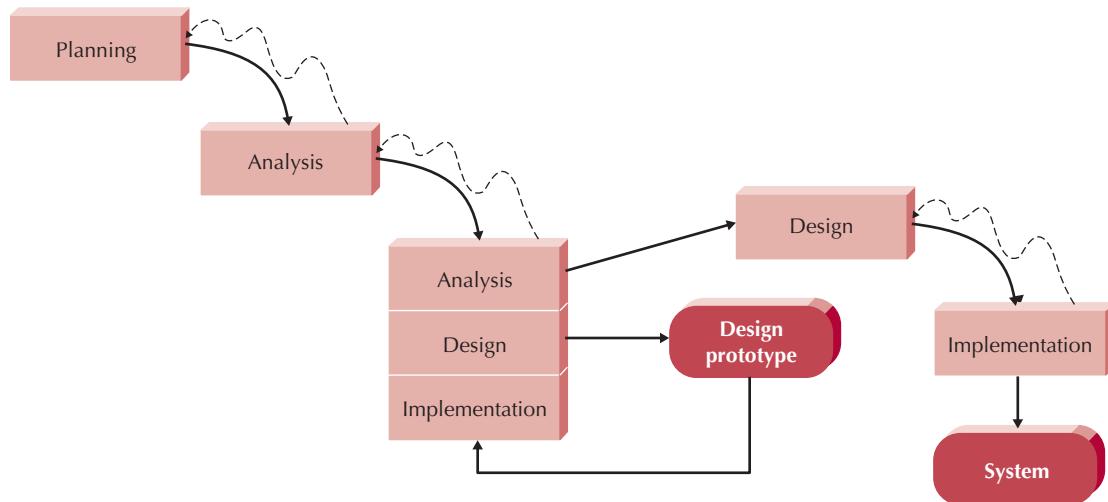


FIGURE 1-6 A Throwaway Prototyping-based Methodology

a Web browser to help the users visualize such a system. In this case, a series of mock-up screens *appear* to be a system, but they really do nothing. Or, suppose that the project team needs to develop a sophisticated graphics program in Java. The team could write a portion of the program with pretend data to ensure that they could do a full-blown program successfully.

A system that is developed using this type of methodology probably relies on several design prototypes during the analysis and design phases. Each of the prototypes is used to minimize the risk associated with the system by confirming that important issues are understood before the real system is built. Once the issues are resolved, the project moves into design and implementation. At this point, the design prototypes are thrown away, which is an important difference between these methodologies and prototyping methodologies, in which the prototypes evolve into the final system.

Throwaway prototyping-based methodologies balance the benefits of well-thought-out analysis and design phases with the advantages of using prototypes to refine key issues before a system is built. It may take longer to deliver the final system as compared to prototyping-based methodologies (since the prototypes do not become the final system), but this type of methodology usually produces more stable and reliable systems.

Agile Development⁶

A third category of systems development methodologies is still emerging today: *agile development*.⁷ These programming-centric methodologies have few rules and practices, all of which are fairly easy to follow. They focus on streamlining the SDLC by eliminating much of the modeling and documentation overhead, and the time spent on those tasks. Instead, projects emphasize simple, iterative application development. Examples

⁶ Two good sources of information on agile development and object-oriented systems is S.W. Ambler, *Agile Modeling: Effective Practices for Extreme Programming and The Unified Process* (New York: John Wiley & Sons, 2002), and R.C. Martin, *Agile Software Development: Principles, Patterns, and Practices* (Englewood Cliffs, NJ: Prentice Hall, 2003).

⁷ For more information, see www.AgileAlliance.org.

of agile development methodologies include extreme programming (XP),⁸ Scrum,⁹ and the Dynamic Systems Development Method (DSDM).¹⁰ Next, we describe XP as an example agile development approach. Typically, XP is used in conjunction with object-oriented methodologies.

Extreme Programming¹¹ *Extreme programming* (XP) is founded on four core values: communication, simplicity, feedback, and courage. These four values provide a foundation on which XP developers use to create any system. First, the developers must provide rapid feedback to the end users on a continuous basis. Second, XP requires developers to follow the KISS principle.¹² Third, developers must make incremental changes to grow the system and they must not only accept change, they must embrace change. Fourth, developers must have a quality first mentality. XP also supports team members in developing their own skills.

Three of the key principles that XP uses to create successful systems are continuous testing, simple coding performed by pairs of developers, and close interactions with end users to build systems very quickly. After a superficial planning process, projects perform analysis, design, and implementation phases iteratively (see Figure 1-7).

Testing and efficient coding practices are core to XP. In fact, each day code is tested and placed into an integrative testing environment. If bugs exist, the code is backed out until it is completely free of errors. XP relies heavily on *refactoring*, which is a disciplined way to restructure code to keep it simple.

An XP project begins with user stories that describe what the system needs to do. Then, programmers code in small, simple modules and test to meet those needs. Users are required to be available to clear up questions and issues as they arise. Standards are very important to minimize confusion, so XP teams use a common set of names, descriptions, and coding practices. XP projects deliver results sooner than even the RAD approaches, and they rarely get bogged down in gathering requirements for the system.

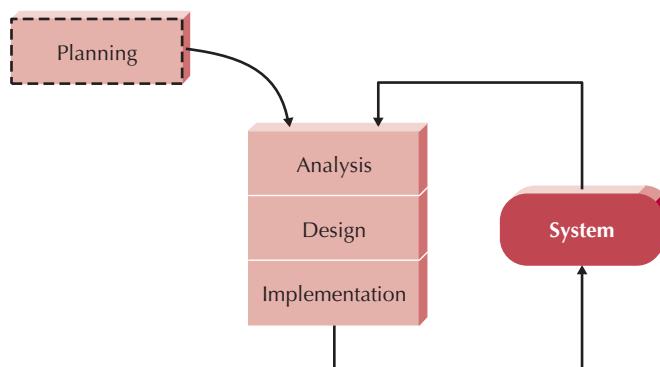


FIGURE 1-7
The Extreme Programming Methodology

⁸ For more information, see www.extremeprogramming.com.

⁹ For more information, see www.controlchaos.com.

¹⁰ For more information, see www.dsmd.org.

¹¹ For more information, see K. Beck, *eXtreme Programming Explained: Embrace Change* (Reading, MA: Addison-Wesley, 2000) and M. Lippert, S. Roock, and H. Wolf, *eXtreme Programming in Action: Practical Experiences from Real World Projects* (New York: John Wiley & Sons, 2002).

¹² Keep It Simple, Stupid.

For small projects with highly motivated, cohesive, stable, and experienced teams, XP should work just fine. However, if the project is not small or the teams aren't jelled¹³ then the success of an XP development effort is doubtful. This tends to throw the whole idea of bringing outside contractors into an existing team environment using XP into doubt.¹⁴ The chance of outsiders "jelling" with insiders may simply be too optimistic. XP requires a great deal of discipline, otherwise projects will become unfocused and chaotic. Furthermore, it is only recommended for small groups of developers—no more than ten developers, and it is not advised for large mission-critical applications. Due to the lack of analysis and design documentation, there is only code documentation associated with XP, maintaining large systems built with XP may be impossible. And since mission-critical business information systems tend to exist for a long time, the utility of XP as a business information system development methodology is in doubt. Finally, the methodology needs a lot of on-site user input, something to which many business units cannot commit.¹⁵

Selecting the Appropriate Development Methodology

Since there are many methodologies, the first challenge faced by analysts is to select which methodology to use. Choosing a methodology is not simple, because no one methodology is always best (if it were, we'd simply use it everywhere!). Many organizations have standards and policies to guide the choice of methodology. You will find that organizations range from having one "approved" methodology, to having several methodology options, to having no formal policies at all.

Figure 1-8 summarizes some important methodology selection criteria. One important item not discussed in this figure is the degree of experience of the analyst team. Many of the RAD-based methodologies require the use of "new" tools and techniques that have a significant learning curve. Often these tools and techniques increase the complexity of the project and require extra time for learning. However, once they are adopted and the team becomes experienced, the tools and techniques can significantly increase the speed in which the methodology can deliver a final system.

Clarity of User Requirements When the user requirements for what the system should do are unclear, it is difficult to understand them by talking about them and explaining them with written reports. Users normally need to interact with technology to really understand what the new system can do and how to best apply it to their needs. Prototyping and throwaway prototyping-based RAD methodologies are usually more appropriate when user requirements are unclear because they provide prototypes for users to interact with early in the SDLC.

Familiarity with Technology When the system will use new technology with which the analysts and programmers are not familiar (e.g., the first Web development project with Java), early application of the new technology in the methodology will improve the chance of success. If the system is designed without some familiarity with the base technology, risks increase because the tools might not be capable of doing what is needed. Throwaway

¹³ A "jelled team" is one that has low turnover, a strong sense of identity, a sense of eliteness, a feeling that they jointly own the product being developed, and enjoyment in working together. For more information regarding jelled teams, see T. DeMarco and T. Lister. *Peopleware: Productive Projects and Teams*, New York, NY (Dorsett, House, 1987).

¹⁴ Considering the tendency for offshore outsourcing, this is a major obstacle for XP to overcome. For more information on offshore outsourcing, see P. Thibodeau, ITAA panel debates outsourcing pros, cons, *Computerworld Morning Update* (September 25, 2003), and S.W. Ambler, "Chicken Little Was Right," *Software Development* (October 2003).

¹⁵ Many of the observations described on the utility of XP as a development approach was based on conversations with Brian Henderson-Sellers.

Ability to Develop Systems	Structured Methodologies			RAD Methodologies		Agile Methodologies
	Waterfall	Parallel	Phased	Prototyping	Throwaway Prototyping	XP
with Unclear User Requirements	Poor	Poor	Good	Excellent	Excellent	Excellent
with Unfamiliar Technology	Poor	Poor	Good	Poor	Excellent	Poor
that are Complex	Good	Good	Good	Poor	Excellent	Poor
that are Reliable	Good	Good	Good	Poor	Excellent	Good
with a Short Time Schedule	Poor	Good	Excellent	Excellent	Good	Excellent
with Schedule Visibility	Poor	Poor	Excellent	Excellent	Good	Good

FIGURE 1-8 Criteria for Selecting a Methodology

prototyping-based methodologies are particularly appropriate for a lack of familiarity with technology because they explicitly encourage the developers to develop design prototypes for areas with high risks. Phased development-based methodologies are good as well, because they create opportunities to investigate the technology in some depth before the design is complete. While one might think prototyping-based methodologies would also be appropriate, they are much less so, since the early prototypes that are built usually only scratch the surface of the new technology. Usually, it is only after several prototypes and several months that the developers discover weaknesses or problems in the new technology.

System Complexity Complex systems require careful and detailed analysis and design. Throwaway prototyping-based methodologies are particularly well suited to such detailed analysis and design, as opposed to prototyping-based methodologies, which are not. The traditional structured design-based methodologies can handle complex systems, but without the ability to get the system or prototypes into the users' hands early on, some key issues may be overlooked. Although phased development-based methodologies enable users to interact with the system early in the process, we have observed that project teams who follow these tend to devote less attention to the analysis of the complete problem domain than they might using others.

System Reliability System reliability is usually an important factor in system development—after all, who wants an unreliable system? However, reliability is just one factor among several. For some applications reliability is truly critical (e.g., medical equipment, missile control systems), while for other applications it is merely important (e.g., games, Internet video). Throwaway prototyping methodologies are the most appropriate when system reliability is a high priority, because it combines detailed analysis and design phases with the ability for the project team to test many different approaches through design prototypes before completing the design. Prototyping methodologies are generally not a good choice when reliability is critical because it lacks the careful analysis and design phases that are essential for dependable systems.

Short Time Schedules Projects that have short time schedules are well suited for RAD-based methodologies. This is due to them being designed to increase the speed of development. Prototyping and phased development-based methodologies are excellent choices when timelines are short because they best enable the project team to adjust the

functionality in the system based on a specific delivery date, and if the project schedule starts to slip, it can be readjusted by removing functionality from the version or prototype under development. Waterfall-based methodologies are the worst choice when time is at a premium because it does not allow for easy schedule changes.

Schedule Visibility One of the greatest challenges in systems development is determining whether a project is on schedule. This is particularly true of the structured design methodologies since design and implementation occur at the end of the project. The RAD-based methodologies move many of the critical design decisions earlier in the project to help project managers recognize and address risk factors and keep expectations in check.

PROJECT TEAM ROLES AND SKILLS

As should be clear from the various phases and steps performed during the SDLC, the project team needs a variety of skills. Project members are *change agents* who identify ways to improve an organization, build an information system to support them, and train and motivate others to use the system. Leading a successful organizational change effort is one of the most difficult jobs that someone can do. Understanding what to change and how to change it, and convincing others of the need for change, requires a wide range of skills. These skills can be broken down into six major categories: technical, business, analytical, interpersonal, management, and ethical.

Analysts must have the technical skills to understand the organization's existing technical environment, the technology that will comprise the new system, and the way in which both can be fit into an integrated technical solution. Business skills are required to understand how IT can be applied to business situations and to ensure that the IT delivers real business value. Analysts are continuous problem solvers at both the project and the organizational level, and they put their analytical skills to the test regularly.

Often, analysts need to communicate effectively one-on-one with users and business managers (who often have little experience with technology) and with programmers (who often have more technical expertise than the analyst). They must be able to give presentations to large and small groups and write reports. Not only do they need to have strong interpersonal abilities, but also they need to manage people with whom they work and they need to manage the pressure and risks associated with unclear situations.

YOUR

TURN

1-1 Selecting a Methodology

Suppose you are an analyst for the Roanoke Software Consulting Company (RSCC), a large consulting firm with offices around the world. The company wants to build a new knowledge management system that can identify and track the expertise of individual consultants anywhere in the world based on their education and the various consulting projects on which they have worked. Assume that this is a new idea that has never before been attempted in RSCC or elsewhere. RSCC has an

international network, but the offices in each country may use somewhat different hardware and software. RSCC management wants the system up and running within a year.

Question:

1. What type of methodology would you recommend RSCC use? Why?

Finally, analysts must deal fairly, honestly, and ethically with other project team members, managers, and system users. Analysts often deal with confidential information or information that, if shared with others, could cause harm (e.g., dissent among employees); it is important to maintain confidence and trust with all people.

In addition to these six general skill sets, analysts require many specific skills that are associated with roles that are performed on a project. In the early days of systems development, most organizations expected one person, the analyst, to have all of the specific skills needed to conduct a systems development project. Some small organizations still expect one person to perform many roles, but because organizations and technology have become more complex, most large organizations now build project teams that contain several individuals with clearly defined responsibilities. Different organizations divide the roles differently, but Figure 1-9 presents one commonly used set of project team roles. Most IS teams include many other individuals, such as the *programmers*, who actually write the programs that make up the system, *network engineers*, who focus on the design of the network, *database administrators*, who deal with optimizing the physical design of the database, and *technical writers*, who prepare the help screens and other documentation (e.g., users manuals, systems manuals).

Business Analyst

The *business analyst* focuses on the business issues surrounding the system. These include identifying the business value that the system will create, developing ideas and suggestions for how the business processes can be improved, and designing the new processes and policies in conjunction with the systems analyst. This individual will likely have business experience and some type of professional training (e.g., the business analyst for accounting systems will likely be a CPA [in the United States] or a CA [in Canada]). He or she represents the interests of the project sponsor and the ultimate users of the system. The business analyst assists in the planning and design phases, but is most active in the analysis phase.

Role	Responsibilities
Business analyst	Analyzing the key business aspects of the system Identifying how the system will provide business value Designing the new business processes and policies
Systems analyst	Identifying how technology can improve business processes Designing the new business processes Designing the information system Ensuring that the system conforms to information systems standards
Infrastructure analyst	Ensuring the system conforms to infrastructure standards Identifying infrastructure changes needed to support the system
Change management analyst	Developing and executing a change management plan Developing and executing a user training plan
Project manager	Managing the team of analysts, programmers, technical writers, and other specialists Developing and monitoring the project plan Assigning resources Serving as the primary point of contact for the project

FIGURE 1-9
Project Team Roles

Systems Analyst

The *systems analyst* focuses on the IS issues surrounding the system. This person develops ideas and suggestions for how information technology can improve business processes, designs the new business processes with help from the business analyst, designs the new information system, and ensures that all IS standards are maintained. The systems analyst likely will have significant training and experience in analysis and design, programming, and even areas of the business. He or she represents the interests of the IS department and works intensively through the project, but perhaps less so during the implementation phase.

Infrastructure Analyst

The *infrastructure analyst* focuses on the technical issues surrounding how the system will interact with the organization's technical infrastructure (e.g., hardware, software, networks and databases). The infrastructure analyst tasks include ensuring that the new information system conforms to organizational standards and identifying infrastructure changes needed to support the system. This individual will likely have significant training and experience in networking, database administration, and various hardware and software products. He or she represents the interests of the organization and IS group that will ultimately have to operate and support the new system once it has been installed. The infrastructure analyst works throughout the project, but perhaps less so during planning and analysis phases.

Change Management Analyst

The *change management analyst* focuses on the people and management issues surrounding the system installation. The roles of this person include ensuring that the adequate documentation and support are available to users, providing user training on the new system, and developing strategies to overcome resistance to change. This individual likely will have significant training and experience in organizational behavior in general, and change management in particular. He or she represents the interests of the project sponsor and users for whom the system is being designed. The change management analyst works most actively during the implementation phase, but begins laying the groundwork for change during the analysis and design phases.

Project Manager

The *project manager* is responsible for ensuring that the project is completed on time and within budget and that the system delivers all benefits that were intended by the project sponsor. The role of the project manager includes managing the team members, developing the project plan, assigning resources, and being the primary point of contact when people outside the team have questions about the project. This individual likely will have significant experience in project management and likely has worked for many years as a systems analyst beforehand. He or she represents the interests of the IS department and the project sponsor. The project manager works intensely during all phases of the project.

**YOUR
TURN**

1.2 Being an Analyst

Suppose you decide to become an analyst after you graduate. What type of analyst would you most prefer to be? What type of courses should you take before you graduate? What type of summer job or internship should you seek?

Question:

1. Develop a short plan that describes how you will prepare for your career as an analyst.

SUMMARY

The Systems Development Life Cycle

All systems development projects follow essentially the same fundamental process called the system development life cycle (SDLC). SLDC starts with a planning phase in which the project team identifies the business value of the system, conducts a feasibility analysis, and plans the project. The second phase is the analysis phase, in which the team develops an analysis strategy, gathers information, and builds a set of analysis models. In the next phase, the design phase, the team develops the physical architecture design, interface design, database and file specifications, and program design. In the final phase, implementation, the system is built, installed, and maintained.

The Evolution of Systems Development Methodologies

System development methodologies are formalized approaches to implementing an SDLC. System development methodologies have evolved over the decades. Structured design methodologies, such as waterfall and parallel development, emphasize decomposition of a problem by either focusing on process decomposition (process-centric methodologies) or data decomposition (data-centric methodologies). They produce a solid, well-thought-out system, but can overlook requirements because users must specify them early in the design process before seeing the actual system. RAD-based methodologies attempt to speed up development and make it easier for users to specify requirements by having parts of the system developed sooner either by producing different versions (phased development) or by using prototypes (prototyping, throwaway prototyping) through the use of CASE tools and fourth generation/visual programming languages. However, RAD-based methodologies still tend to be either process-centric or data-centric. Agile development methodologies, such as XP, focus on streamlining the SDLC by eliminating many of the tasks and time associated with requirements definition and documentation. Several factors influence the choice of a methodology: clarity of the user requirements; familiarity with the base technology; system complexity; need for system reliability; time pressures; and the need to see progress on the time schedule.

Project Team Roles and Skills The project team needs a variety of skills. All analysts need to have general skills, such as change management, ethics, communications, and technical. However, different kinds of analysts require specific skills in addition to these. Business analysts usually have business skills that help them to understand how the business issues surrounding the system, whereas systems analysts also have significant experience in analysis and design and programming. The infrastructure analyst focuses on technical issues surrounding how the system will interact with the organization's technical infrastructure, and the change management analyst focuses on people and management issues surrounding the system installation. In addition to analysts, project teams will include a project manager, programmers, technical writers, and other specialists.

KEY TERMS

Agile development	Implementation phase	Rapid application development (RAD)
Analysis model	Infrastructure analyst	Requirements gathering
Analysis phase	Installation	Steering committee
Analysis strategy	Interface design	Step
Approval committee	Methodology	Structured design
Architecture design	Network engineer	Support plan
As-is system	Object-oriented methodology	System development life cycle (SDLC)
Business analyst	Parallel development	System proposal
Change agent	Phase	System prototype
Change management analyst	Phased development	System request
Construction	Planning phase	System specification
Database administrator	Process-centered methodology	Systems analyst
Database and file specification	Program design	Technical writer
Data-centered methodology	Programmer	Technique
Deliverable	Project initiation	Throwaway prototyping
Design phase	Project management	To-be system
Design prototype	Project manager	Training plan
Design strategy	Project plan	Version
Extreme programming (XP)	Project sponsor	Waterfall development
Feasibility analysis	Prototyping	Workplan
Gradual refinement	Refactoring	

QUESTIONS

1. Compare and contrast phases, steps, techniques, and deliverables.
2. Describe the major phases in the systems development life cycle (SDLC).
3. Describe the principal steps in the planning phase. What are the major deliverables?
4. Describe the principal steps in the analysis phase. What are the major deliverables?
5. Describe the principal steps in the design phase. What are the major deliverables?
6. Describe the principal steps in the implementation phase. What are the major deliverables?
7. What are the roles of a project sponsor and the approval committee?
8. What does gradual refinement mean in the context of SDLC?
9. Compare and contrast process-centered methodologies with data-centered methodologies.
10. Compare and contrast structured design-based methodologies in general to RAD-based methodologies in general.
11. Compare and contrast extreme programming and throwaway prototyping.
12. Describe the major elements and issues with waterfall development.
13. Describe the major elements and issues with parallel development.
14. Describe the major elements and issues with phased development.
15. Describe the major elements and issues with prototyping.
16. Describe the major elements and issues with throwaway prototyping.
17. What are the key factors in selecting a methodology?
18. What are the major roles on a project team?
19. Compare and contrast the role of a systems analyst, business analyst, and infrastructure analyst.
20. Which phase in the SDLC is most important? Why?

EXERCISES

- A. Suppose you are a project manager using a waterfall development-based methodology on a large and complex project. Your manager has just read the latest article in *Computerworld* that advocates replacing this methodology with prototyping and comes to your office requesting you to switch. What do you say?

Retrieved from: www.knovel.com

- B. The basic types of methodologies discussed in this chapter can be combined and integrated to form new hybrid methodologies. Suppose you were to combine throw-away prototyping with the use of waterfall development. What would the methodology look like? Draw a picture (similar to those in Figures 1-2 through 1-7). How would this new methodology compare to the others?
- C. Suppose you were an analyst working for a small company to develop an accounting system. What type of methodology would you use? Why?
- D. Suppose you were an analyst developing a new executive information system intended to provide key strategic information from existing corporate databases to senior executives to help in their decision making. What type of methodology would you use? Why?
- E. Suppose you were an analyst developing a new information system to automate the sales transactions and manage inventory for each retail store in a large chain. The system would be installed at each store and exchange data with a mainframe computer at the company's head office. What type of methodology would you use? Why?
- F. Look in the classified section of your local newspaper. What kinds of job opportunities are available for people who want analyst positions? Compare and contrast the skills that the ads ask for to the skills that we presented in this chapter.
- G. Think about your ideal analyst position. Write a newspaper ad to hire someone for that position. What requirements would the job have? What skills and experience would be required? How would an applicant be able to demonstrate having the appropriate skills and experience?

MINICASES

1. Barbara Singleton, manager of western regional sales at the WAMAP Company, requested that the IS department develop a sales force management and tracking system that would enable her to better monitor the performance of her sales staff. Unfortunately, due to the massive backlog of work facing the IS department, her request was given a low priority. After six months of inaction by the IS department, Barbara decided to take matters into her own hands. Based on the advice of friends, Barbara purchased a PC and simple database software and constructed a sales force management and tracking system on her own.

Although Barbara's system has been "completed" for about six weeks, it still has many features that do not work correctly, and some functions are full of errors. Barbara's assistant is so mistrustful of the system that she has secretly gone back to using her old paper-based system, since it is much more reliable.

Over dinner one evening, Barbara complained to a systems analyst friend, "I don't know what went wrong with this project. It seemed pretty simple to me. Those IS guys wanted me to follow this elaborate set of steps and tasks, but I didn't think all that really applied to a PC-based system. I just thought I could build this system and tweak it around until I got what I wanted without all the fuss and bother of the methodology the IS guys were pushing. I mean, doesn't that just apply to their big, expensive systems?"

Assuming you are Barbara's systems analyst friend, how would you respond to her complaint?

2. Marcus Weber, IS project manager at ICAN Mutual Insurance Co., is reviewing the staffing arrangements for his next major project, the development of an expert

system-based underwriters assistant. This new system will involve a whole new way for the underwriters to perform their tasks. The underwriters assistant system will function as sort of an underwriting supervisor, reviewing key elements of each application, checking for consistency in the underwriter's decisions, and ensuring that no critical factors have been overlooked. The goal of the new system is to improve the quality of the underwriters' decisions and to improve underwriter productivity. It is expected that the new system will substantially change the way the underwriting staff do their jobs.

Marcus is dismayed to learn that due to budget constraints, he must choose between one of two available staff members. Barry Filmore has had considerable experience and training in individual and organizational behavior. Barry has worked on several other projects in which the end users had to make significant adjustments to the new system, and Barry seems to have a knack for anticipating problems and smoothing the transition to a new work environment. Marcus had hoped to have Barry's involvement in this project.

Marcus's other potential staff member is Kim Danville. Prior to joining ICAN Mutual, Kim had considerable work experience with the expert system technologies that ICAN has chosen for this expert system project. Marcus was counting on Kim to help integrate the new expert system technology into ICAN's systems environment, and also to provide on-the-job training and insights to the other developers on this team.

Given that Marcus's budget will only permit him to add Barry or Kim to this project team, but not both, what choice do you recommend for him? Justify your answer.

CHAPTER 2

INTRODUCTION TO OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN WITH THE UNIFIED MODELING LANGUAGE, VERSION 2.0

This chapter introduces Object-Oriented Systems Analysis and Design with the Unified Modeling Language, Version 2.0. First, the chapter introduces the basic characteristics of object-oriented systems. Second, it introduces UML 2.0. Third, the chapter overviews Object-Oriented Systems Analysis and Design and describes the Unified Process. Finally, based on the Unified Process and the UML 2.0, the chapter provides a minimalist approach to Object-Oriented Systems Analysis and Design with UML 2.0.

OBJECTIVES:

- Understand the basic characteristics of object-oriented systems.
- Be familiar with the Unified Modeling Language (UML), Version 2.0.
- Be familiar with the Unified Process.
- Understand a minimalist approach to object-oriented systems analysis and design.

CHAPTER OUTLINE

Introduction	Use-case driven
Basic Characteristics of Object-Oriented Systems	Architecture Centric
Classes and Objects	Iterative and Incremental
Methods and Messages	The Unified Process
Encapsulation and Information Hiding	A Minimalist Approach to Object-Oriented
Inheritance	Systems Analysis and Design with UML 2.0
Polymorphism and Dynamic Binding	Benefits of Object-Oriented Systems
The Unified Modeling Language, Version 2.0	Analysis and Design
Structure Diagrams	Extensions to the Unified Process
Behavior Diagrams	The Minimalist Object-Oriented
Extension Mechanisms	Systems Analysis and Design Approach
Object-Oriented Systems Analysis and Design	Summary

INTRODUCTION

Until recent years, analysts focused on either data or business processes when developing systems. As they moved through the SDLC, they emphasized either the data for the system (data-centric approaches) or the processes that it would support (process-centric approaches). In the mid-1980s, developers were capable of building systems that could be more efficient if the analyst worked with a system's data and processes simultaneously and focused on integrating the two. As such, project teams began using an *object-oriented approach*, whereby self-contained modules called *objects* (containing both data and processes) were used as the building blocks for systems.

The ideas behind object-oriented approaches are not new. They can be traced back to the object-oriented programming languages *Simula*, created in the 1960s, and *Smalltalk*, created in the early 1970s. Until the mid-1980s, developers had to keep the data and processes separate to be capable of building systems that could run on the mainframe computers of that era. Today, due to the increase in processor power and the decrease in processor cost, object-oriented approaches are feasible. One of the major hurdles of learning object-oriented approaches to developing information systems is the volume of new terminology. In this chapter we overview the basic characteristics of an object-oriented system, provide an overview of the second version of the Unified Modeling Language (UML, 2.0), introduce basic object-oriented systems analysis and design and the Unified Process, and present a minimalist approach to object-oriented systems analysis and design with UML 2.0.

BASIC CHARACTERISTICS OF OBJECT-ORIENTED SYSTEMS

Object-oriented systems focus on capturing the structure and behavior of information systems in little modules that encompass both data and process. These little modules are known as objects. In this section of the chapter we describe the basic characteristics of object-oriented systems, which include classes, objects, methods, messages, encapsulation, information hiding, inheritance, polymorphism, and dynamic binding.

Classes and Objects

A *class* is the general template we use to define and create specific instances, or objects. Every object is associated with a class. For example, all of the objects that capture information about patients could fall into a class called Patient, because there are attributes (e.g., names, addresses, and birth dates) and methods (e.g., insert new instances, maintain information, and delete entries) that all patients share (see Figure 2-1).

An *object* is an instantiation of a class. In other words, an object is a person, place, event, or thing about which we want to capture information. If we were building an appointment system for a doctor's office, classes might include doctor, patient, and appointment. The specific patients like Jim Maloney, Mary Wilson, and Theresa Marks are considered *instances*, or objects, of the patient class.

Each object has *attributes* that describe information about the object, such as a patient's name, birth date, address, and phone number. The *state* of an object is defined by the value of its attributes and its relationships with other objects at a particular point in time. For example, a patient might have a state of "new" or "current" or "former."

Each object also has *behaviors*. The behaviors specify what the object can do. For example, an appointment object likely can schedule a new appointment, delete an appointment, and locate the next available appointment.

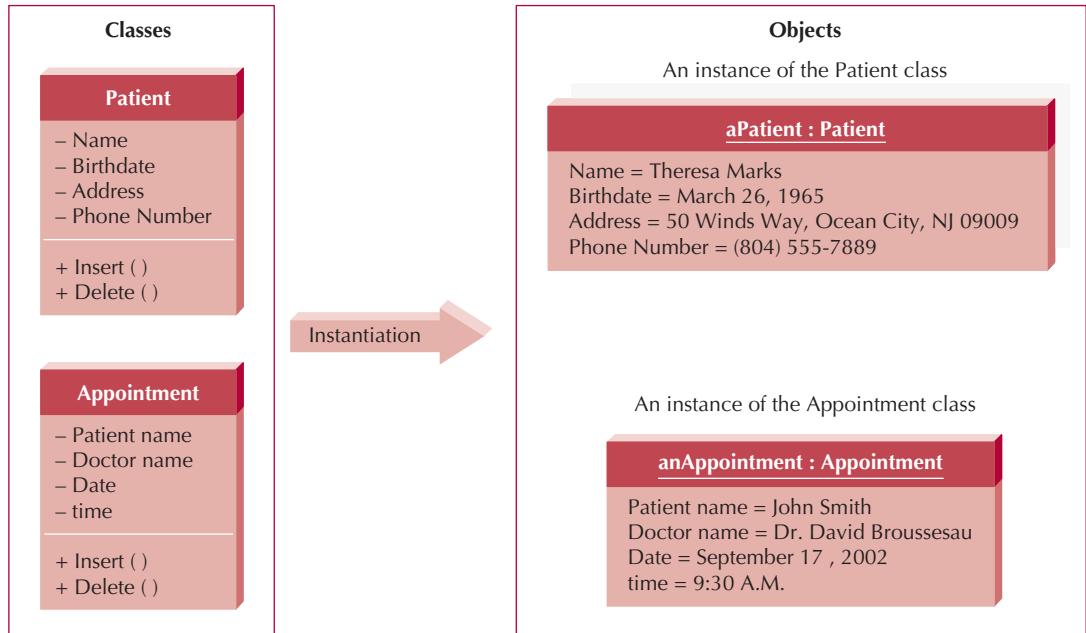


FIGURE 2-1 Classes and Objects

One of the more confusing aspects of object-oriented systems development is the fact that in most object-oriented programming languages, both classes and instances of classes can have attributes and methods. Class attributes and methods tend to be used to model attributes (or methods) that deal with issues related to all instances of the class. For example, to create a new patient object, a message is sent to the patient class to create a new instance of itself. However, from a systems analysis and design point of view, we will focus primarily on attributes and methods of objects and not of classes.

Methods and Messages

Methods implement an object's behavior. A method is nothing more than an action that an object can perform. As such, they are analogous to a function or procedure in a traditional programming language such as C, Cobol, or Pascal. *Messages* are information sent to objects to trigger methods. Essentially, a message is a function or procedure call from one object to another object. For example, if a patient is new to the doctor's office, the system will send an insert message to the application. The patient object will receive a message (instruction) and do what it needs to do to go about inserting the new patient into the system (execute its method). See Figure 2-2.

Encapsulation and Information Hiding

The ideas of *encapsulation* and *information hiding* are interrelated in object-oriented systems. However, neither of the terms is new. Encapsulation is simply the combination of process and data into a single entity. Traditional approaches to information systems development tend to be either process-centric (e.g., structured systems) or data-centric (e.g., information engineering). Object-oriented approaches combine process and data into holistic entities (objects).

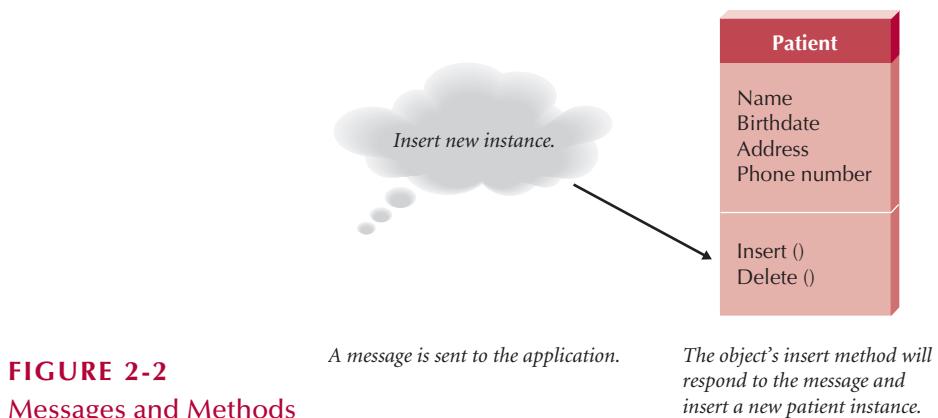


FIGURE 2-2
Messages and Methods

Information hiding was first promoted in structured systems development. The principle of information hiding suggests that only the information required to use a software module be published to the user of the module. Typically, this implies the information required to be passed to the module, and the information returned from the module is published. Exactly how the module implements the required functionality is not relevant. We really do not care how the object performs its functions, as long as the functions occur. In object-oriented systems, combining encapsulation with the information hiding principle suggests that the information hiding principle be applied to objects instead of merely applying it to functions or processes. As such, objects are treated like black boxes.

The fact that we can use an object by calling methods is the key to reusability because it shields the internal workings of the object from changes in the outside system, and it keeps the system from being affected when changes are made to an object. In Figure 2-2, notice how a message (insert new patient) is sent to an object yet the internal algorithms needed to respond to the message are hidden from other parts of the system. The only information that an object needs to know is the set of operations, or methods, that other objects can perform and what messages need to be sent to trigger them.

Inheritance

Inheritance, as an information systems development characteristic, was proposed in data modeling in the late 1970s and the early 1980s. The data modeling literature suggests using inheritance to identify higher-level, or more general, classes of objects. Common sets of attributes and methods can be organized into *superclasses*. Typically, classes are arranged in a hierarchy whereby the *superclasses*, or general classes, are at the top, and the *subclasses*, or specific classes, are at the bottom. In Figure 2-3, person is a superclass to the classes Doctor and Patient. Doctor, in turn, is a superclass to general practitioner and specialist. Notice how a class (e.g., doctor) can serve as a superclass and subclass concurrently. The relationship between the class and its superclass is known as the *A-Kind-Of* (AKO) relationship. For example, in Figure 2-3, a general practitioner is A-Kind-Of doctor, which is A-Kind-Of person.

Subclasses *inherit* the appropriate attributes and methods from the superclasses above them. That is, each subclass contains attributes and methods from its parent superclass. For example, Figure 2-3 shows that both doctor and patient are subclasses of person and therefore will inherit the attributes and methods of the person class. Inheritance makes it simpler to define classes. Instead of repeating the attributes and methods in the doctor and patient classes separately, the attributes and methods that are common

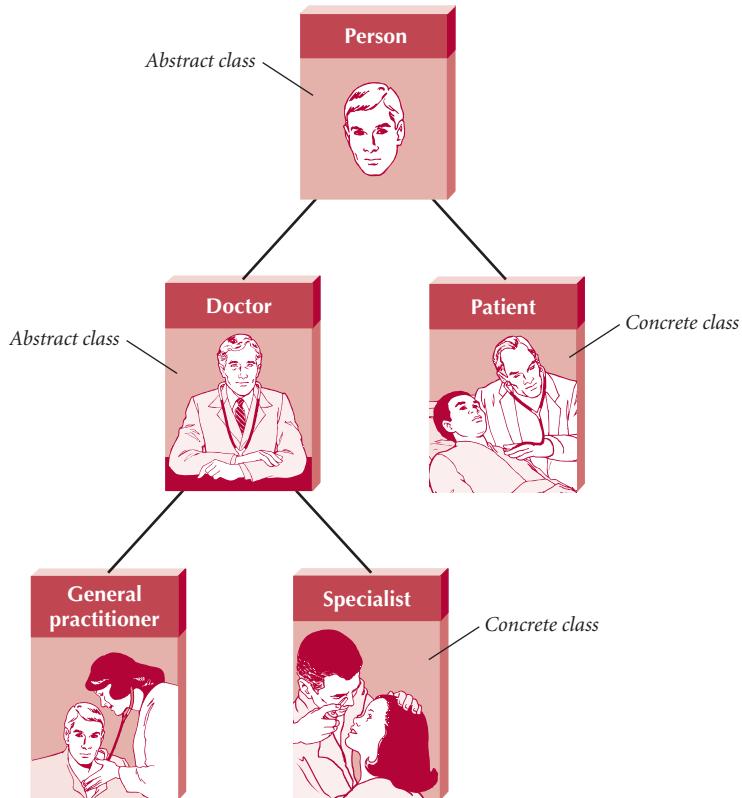


FIGURE 2-3
Class Hierarchy

to both are placed in the person class and inherited by those classes below it. Notice how much more efficient hierarchies of object classes are than the same objects without a hierarchy in Figure 2-4.

Most classes throughout a hierarchy will lead to instances; any class that has instances is called a *concrete class*. For example, if Mary Wilson and Jim Maloney were instances of the patient class, patient would be considered a concrete class. Some classes do not produce instances because they are used merely as templates for other more specific classes (especially those classes located high up in a hierarchy). The classes are referred to as *abstract classes*. Person would be an example of an abstract class. Instead of creating objects from person, we would create instances representing the more specific classes of Doctor and Patient, both types of *person* (see Figure 2-3). What kind of class is the general practitioner class? Why?

**YOUR
TURN**

2-1 Encapsulation and Information Hiding

Come up with a set of examples of using encapsulation and information hiding in everyday life. For example, is there any information about yourself that you would not mind if everyone knew? How would someone retrieve

this information? What about personal information that you would prefer to be private? How would you prevent someone from retrieving it?

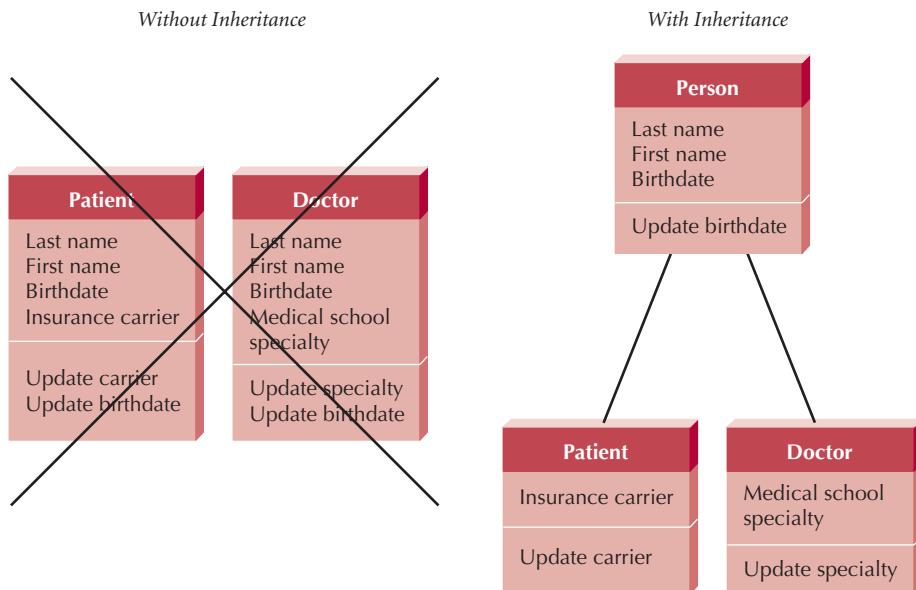


FIGURE 2-4
Inheritance

Polymorphism and Dynamic Binding

Polymorphism means that the same message can be interpreted differently by different classes of objects. For example, inserting a patient means something different than inserting an appointment. As such, different pieces of information need to be collected and stored. Luckily, we do not have to be concerned with *how* something is done when using objects. We can simply send a message to an object, and that object will be responsible for interpreting the message appropriately. For example, if we sent the message “Draw yourself” to a square object, a circle object, and a triangle object, the results would be very different, even though the message is the same. Notice in Figure 2-5 how each object responds appropriately (and differently) even though the messages are identical.

Polymorphism is made possible through *dynamic binding*. Dynamic, or late, binding is a technique that delays typing the object until run-time. As such, the specific method that is actually called is not chosen by the object-oriented system until the system is running. This is in contrast to *static binding*. In a statically bound system, the type of object would be determined at compile time. Therefore, the developer would have to choose which method should be called instead of allowing the system to do it. This is why in most traditional programming languages you find complicated decision logic based on the different types of objects in a system. For example, in a traditional programming language, instead of sending the message “Draw yourself” to the different

**YOUR
TURN**

2-2 Inheritance

See if you can come up with at least three different classes that you might find in a typical business situation. Select one of the classes and create at least a

three-level inheritance hierarchy using the class. Which of the classes are abstract, if any, and which ones are concrete?

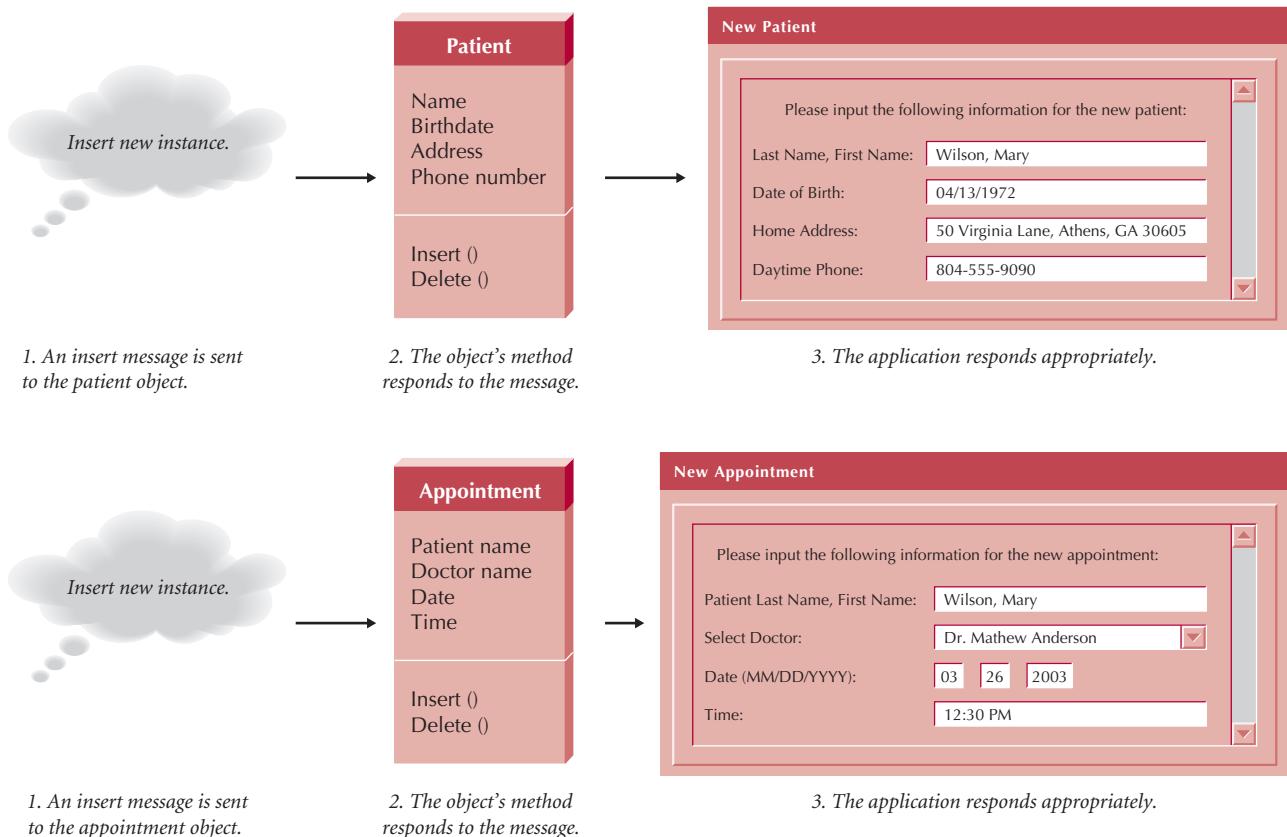


FIGURE 2-5 Polymorphism and Encapsulation

types of graphical objects in Figure 2-5, you would have to write decision logic using a case statement or a set of if statements to determine what kind of graphical object you wanted to draw, and you would have to name each draw function differently (e.g., draw-square, draw-circle, or draw-triangle). This obviously would make the system much more complicated and more difficult to understand.

THE UNIFIED MODELING LANGUAGE, VERSION 2.0

Until 1995, object concepts were popular but implemented in many different ways by different developers. Each developer had his or her own methodology and notation (e.g., Booch, Coad, Moses, OMT, OOSE, and SOMA¹). Then in 1995, Rational Software brought three industry leaders together to create a single approach to object-oriented sys-

¹ See Grady Booch, *Object-Oriented Analysis and Design with Applications*, 2nd Ed. (Redwood City, CA: Benjamin/Cummings, 1994); Peter Coad and Edward Yourdon, *Object-Oriented Analysis*, 2nd Ed. (Englewood Cliffs, NJ: Yourdon Press, 1991); Peter Coad and Edward Yourdon, *Object-Oriented Design* (Englewood Cliffs, NJ: Yourdon Press, 1991); Brian Henderson-Sellers and Julian Edwards, *BookTwo of Object-Oriented Knowledge: The Working Object* (Sydney, Australia: Prentice Hall, 1994); James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, *Object-Oriented Modeling and Design* (Englewood Cliffs, NJ, 1991); Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Overgaard, *Object-Oriented Software Engineering: A Use Case Approach* (Wokingham, England: Addison-Wesley, 1992); Ian Graham, *Migrating to Object Technology* (Wokingham, England: Addison-Wesley, 1994).

**YOUR
TURN**

2-3 Polymorphism and Dynamic Binding

Can you think of any way in which you use polymorphism and/or dynamic binding in your everyday life? For example, when you are told to do some task, do you always perform the task like everyone else you know

does? Do you always perform the task the same way or is the method of performance depend on where you are when you perform the task?

tems development. Grady Booch, Ivar Jacobson, and James Rumbaugh worked with others to create a standard set of diagramming techniques known as the *Unified Modeling Language (UML)*. The objective of UML is to provide a common vocabulary of object-oriented terms and diagramming techniques that is rich enough to model any systems development project from analysis through implementation. In November 1997, the *Object Management Group (OMG)* formally accepted UML as the standard for all object developers. Over the years since, the UML has gone through multiple minor revisions. The current version of UML, Version 2.0, was accepted by the members of the OMG during their spring and summer meetings of 2003.

The Version 2.0 of the UML defines a set of fourteen diagramming techniques used to model a system. The diagrams are broken into two major groupings: one for modeling structure of a system and one for modeling behavior. The structure modeling diagrams include class, object, package, deployment, component, and composite structure diagrams. The behavior modeling diagrams include activity, sequence, communication, interaction overview, timing, behavior state machine, protocol state machine, and use case diagrams.² Figure 2-6 provides and overview of these diagrams.

Depending on where in the development process the system is, different diagrams play a more important role. In some cases, the same diagramming technique is used throughout the development process. In that case, the diagrams start off very conceptual and abstract. As the system is developed, the diagrams evolve to include details that ultimately lead to code generation and development. In other words, the diagrams move from documenting the requirements to laying out the design. Overall, the consistent notation, integration among the diagramming techniques, and the application of the diagrams across the entire development process makes the UML a powerful and flexible language for analysts and developers. In the remainder of this section, we provide an overview of the diagramming techniques supported by the UML. In later chapters, we provide more detail on using a subset of the UML in object-oriented systems analysis and design.

Structure Diagrams

In this section of the chapter, we introduce the static, *structure diagrams* of the UML 2.0. As mentioned above, the structure diagrams include the class, object, package, deployment, component, and composite structure diagrams. Structure diagrams provide a way for representing the data and static relationships that are in an information system. Below, we describe the basic purpose of each of the structure diagrams.

² The material contained in this section is based on the *Unified Modeling Language: Superstructure Version 2.0, ptc/03-08-02* (www.uml.org). Additional useful references include Michael Jesse Chonoles and James A. Schardt, *UML 2 for Dummies* (Indianapolis, IN: Wiley, 2003), Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado, *UML 2 Toolkit* (Indianapolis, IN: Wiley, 2004), and Kendall Scott, *Fast Track UML 2.0* (Berkeley, CA: Apress, 2004). For a complete description of all diagrams, see www.uml.org.

Diagram Name	Used to	Primary Phase
Structure Diagrams		
Class	Illustrate the relationships between classes modeled in the system.	Analysis, Design
Object	Illustrate the relationships between objects modeled in the system.	Analysis, Design
Package	Used when actual instances of the classes will better communicate the model. Group other UML elements together to form higher level constructs.	Analysis, Design, Implementation
Deployment	Show the physical architecture of the system. Can also be used to show software components being deployed onto the physical architecture.	Physical Design, Implementation
Component	Illustrate the physical relationships among the software components.	Physical Design, Implementation
Composite Structure	Illustrate the internal structure of a class, i.e., the relationships among the parts of a class.	Analysis, Design
Behavioral Diagrams		
Activity	Illustrate business workflows independent of classes, the flow of activities in a use case, or detailed design of a method.	Analysis, Design
Sequence	Model the behavior of objects within a use case. Focuses on the time-based ordering of an activity.	Analysis, Design
Communication	Model the behavior of objects within a use case. Focuses on the communication among a set of collaborating objects of an activity.	Analysis, Design
Interaction Overview	Illustrate an overview of the flow of control of a process.	Analysis, Design
Timing	Illustrate the interaction that takes place among a set of objects and the state changes in which they go through along a time axis.	Analysis, Design
Behavioral State Machine	Examine the behavior of one class.	Analysis, Design
Protocol State Machine	Illustrates the dependencies among the different interfaces of a class.	Analysis, Design
Use-Case	Capture business requirements for the system and to illustrate the interaction between the system and its environment.	Analysis

FIGURE 2-6 UML 2.0 Diagram Summary

Class Diagrams The primary purpose of the class diagram is to create a vocabulary that is used by both the analyst and users. *Class diagrams* typically represent the things, ideas or concepts that are contained in the application. For example, if you were building a payroll application, a class diagram would probably contain classes that represent things such as employees, checks, and the payroll register. The class diagram would also portray the relationships among classes. The actual syntax of the class diagram is presented in Chapter 7.

Object Diagrams *Object diagrams* are very similar to class diagrams. The primary difference is that an object diagram portrays objects and their relationships. The primary purpose of an object diagram is to allow an analyst to uncover additional details of a class. In some cases, instantiating a class diagram may aid a user or analyst in discovering additional relevant attributes, relationships, and/or operations, or possibly discover that some of the attributes, relationships, or operations have been misplaced. Like the class diagram, the actual syntax and use of the object diagram is presented in Chapter 7.

Package Diagrams *Package diagrams* are primarily used to group elements of the other UML diagrams together into a higher-level construct: a *package*. Package diagrams are essentially class diagrams that only show packages, instead of classes, and dependency relationships, instead of the typical relationships shown on class diagrams. For example, if we had an appointment system for a doctor's office, it may make sense to group a patient class with the patient's medical history class together to form a patient class package.

Furthermore, it could be useful to create a treatment package that contains symptoms of illnesses, illnesses, and the typical medications that are prescribed for them. We provide more details in using package diagrams in Chapters 6 through 9.

Deployment Diagrams *Deployment diagrams* are used to represent the relationships between the hardware components used in the physical infrastructure of an information system. For example, when designing a distributed information system that will use a wide area network, a deployment diagram can be used to show the communication relationships among the different nodes in the network. They also can be used to represent the software components and how they are deployed over the physical architecture or infrastructure of an information system. In this case, a deployment diagram represents the environment for the execution of the software. In Chapter 13, we describe designing the physical architecture of an information system and use an extension to the deployment diagram to represent the design.

Component Diagrams *Component diagrams* allow the designer to model physical relationships among the physical modules of code. The diagram when combined with the deployment diagram can be used to portray the physical distribution of the software modules over a network. For example, when designing client-server systems, it is useful to show which classes or packages of classes will reside on the client nodes and which ones will reside on the server. Component diagrams also can be useful in designing and developing component-based systems. Since this book focuses on object-oriented systems analysis and design, we will not discuss further the use of component diagrams.

Composite Structure Diagrams The UML 2.0 provides a new diagram for when the internal structure of a class is complex: the composite structure diagram. *Composite structure diagrams* are used to model the relationships among parts of a class. For example, when modeling a payroll register, an analyst may want a class that represents the entire report as well as classes that represent the header, footer, and detail lines of the report. In a standard class diagram, this would require the analyst to model the payroll register as four separate classes with relationships connecting them together. Instead, the composite structure diagram would contain three subclasses: header, footer, and detail lines. Composite structure diagrams also are useful when modeling the internal structure of a component for a component-based system.

Often, the composite structure diagram is a redundant modeling mechanism because its models also can be communicated using packages and package diagrams. Because of this redundancy, and because we are not covering component-based systems development, we will not discuss them further in this book.

YOUR

2-4 Structure Diagrams

TURN

Why are structure diagrams considered to be static? Consider the implementation of a system for the Career Services department of your university that would support the student interview and job placement processes. Briefly describe to your primary contact in the Career Services department the kinds of informa-

tion that the following structure diagrams would communicate: A) class, B) object, C) package, D) deployment, E) component, and F) composite structure. Be sure to include examples from the Career Services department so that your non-technical user will be able to understand your explanations!

Behavior Diagrams

In this section of the chapter, we introduce the dynamic, *behavior diagrams* of the UML 2.0. The behavior diagrams included in UML 2.0 are the activity, sequence, communication, interaction overview, timing, behavior state machine, protocol state machine, and use case diagrams. Behavior modeling diagrams provide the analyst with a way to depict the dynamic relationships among the instances or objects that represent the business information system. They also allow the modeling of the dynamic behavior of individual objects throughout their lifetime. The behavior diagrams support the analyst in modeling the functional requirements of an evolving information system.

Activity Diagrams *Activity diagrams* provide the analyst with the ability to model processes in an information system. Activity diagrams can be used to model workflows, individual use cases, or the decision logic contained within an individual method.³ They also provide an approach to model parallel processes. Activity diagrams are further described in Chapter 6.

Interaction Diagrams *Interaction diagrams* portray the interaction among the objects of an object-oriented information system. UML 2.0 provides four different interaction diagrams: sequence, communication, interaction overview, and timing diagrams. Each of these diagrams is introduced here.

1. Sequence diagrams allow an analyst to portray the dynamic interaction among objects in an information system. Sequence diagrams are by far the most common kind of interaction diagram used in object-oriented modeling. They emphasize the time-based ordering of the activity that takes place with a set of collaborating objects. They are very useful in helping an analyst understand real-time specifications and complex use cases (see below). These diagrams can be used to describe both the logical and physical interactions among the objects. As such, they are useful in both analysis and design activities. We describe sequence diagrams in more detail in Chapter 8.

2. Communication diagrams provide an alternative view of the dynamic interaction that takes place among the objects in an object-oriented information system. Where sequence diagrams emphasize the time-based ordering of an activity, communication diagrams focus on the set of messages that are passed within a set of collaborating objects. In other words, communication diagrams depict how objects collaborate to support some aspect of the required functionality of the system. The sequence or time-based ordering of the messages is shown through a sequence numbering scheme. From a practical point of view, communication diagrams and sequence diagrams provide the same information. As such, we describe communication diagrams in more detail in Chapter 8 with the sequence diagrams.

3. Interaction overview diagrams help analysts understand complex use cases. They provide an overview of a process's flow of control. Interaction overview diagrams extend activity diagrams through the addition of sequence fragments from sequence diagrams. In effect, sequence fragments are treated as if they were activities in an activity diagram.

³ For those who are familiar with traditional structured analysis and design, activity diagrams combine the ideas that underlie data flow diagrams and system flowcharts. Essentially, they are sophisticated and updated data flow diagrams. Technically speaking, activity diagrams combine process modeling ideas from many different techniques including event models, statecharts, and Petri Nets. However, UML 2.0's activity diagram has more in common with Petri Nets than the other process modeling techniques. For a good description of using Petri Nets to model business workflows see Wil van der Aalst and Kees van Hee, *Workflow Management: Models, Methods, and Systems* (Cambridge, MA: MIT Press, 2002).

The primary advantage of using interaction overview diagrams is that you can easily model alternative sequence flows. However, practically speaking, this can be accomplished using activity diagrams and use cases instead. Due to their limited use, interaction overview diagrams are not described in more detail in this book.

4. Timing diagrams portray the interaction between objects along a time axis. The primary purpose of the timing diagram is to show the change of state of an object in response to events over time. They tend to be very useful when developing real-time or embedded systems. However, like interaction overview diagrams, due to their limited use, we do not describe timing diagrams in more detail in this book.

State Machines In UML 2.0, there are two different *state machines*: behavior and protocol.⁴ Behavior state machines are used to describe the changes that an object may go through during its lifetime. Protocol state machines portray a specified sequence of events to which an object may respond.

1. Behavior state machines provide a method for modeling the different states, or sets of values, that instances of a class may go through during their lifetime. For example, a patient can change over time from being a New Patient to a Current Patient to a Former Patient. Each of these “types” of patients is really a different state of the same patient. The different states are connected by events that cause the instance (patient) to transition from one state to another. We describe behavior state machines in more detail in Chapter 8.

2. Protocol state machines support the analyst in designing dependencies among elements of the class’ interface. For example, typically speaking you must open a file or database before querying or updating it. Unlike behavior state machines, protocol state machines may be associated with component ports or class interfaces. Protocol state machines are very specialized. As such, we do not provide any more detail on them in this book.

Use Case Diagrams *Use case diagrams* allow the analyst to model the interaction of an information system and its environment. The environment of an information system includes both the end user and any external system that interacts with the information system. The primary use of the use case diagram is to provide a means to document and understand the requirements of the evolving information system. Use cases and use case diagrams are some of the most important tools that are used in object-oriented systems analysis and design. We describe use cases and use case diagrams in more detail later in this chapter and in Chapter 6.

YOUR

TURN

2-5 Behavior Diagrams

Why are behavior diagrams considered to be static? Consider the implementation of a system for the Career Services department of your university that would support the student interview and job placement processes. Briefly describe to your primary contact in the Career Services department the kinds of information that the following structure diagrams

would communicate: A) activity, B) sequence, C) communication, D) interaction overview, E) timing, F) behavior state machines, G) protocol state machines, and H) use case. Be sure to include examples from the Career Services department so that you non-technical user will be able to understand your explanations!

⁴ UML 2.0 state machines are based on work by David Harel. See David Harel, On Visual Formalisms, *CACM*, 31 (5) (May 1988), 514–530 and David Harel, A Visual Formalism for Complex Systems, *Science of Computer Programming*, 8, (1987), 231–274.

Extension Mechanisms

As large and as complete as the UML is, it is impossible for the creators of the UML to anticipate all potential uses. Fortunately, the creators of the UML also have provided a set of extension mechanisms. These include stereotypes, tagged values, constraints, and profiles. Each of these is described next.

Stereotypes A *stereotype* provides the analyst with the ability to incrementally extend the UML using the model elements already in the UML. A stereotype is shown as a text item enclosed within guillemets (<>) or angle brackets (<<>>). Stereotypes can be associated with any model element (e.g., class, object, use case, relationships) within any UML diagram. We will use stereotypes in Chapter 12 in conjunction with a special form of the behavior state machine: the window navigation diagram.

Tagged Values In the UML, all model elements have properties that describe them. For example, all elements have a name. There are times that it is useful to add properties to the base elements. *Tagged values* are used to add new properties to a base element. For example, if a project team was interested in tracing the authorship of each class in a class diagram, the project team could extend the class element to include an author property. It is also possible to associate tagged values with specific stereotypes. In this manner, when the analyst applies a stereotype to a model element, all of the additional tagged values associated with the stereotype also are applied. We do not describe tagged values in any more detail in this book.

Constraints *Constraints* allow the analyst to model problem domain specific semantics by placing additional restrictions on the use of model elements. Constraints are typically modeled using the Object Constraint Language (OCL).⁵ We return to a discussion of constraints and object-oriented systems analysis and design in Chapter 10.

Profiles *Profiles* allow the developer to group a set of model elements that have been extended using stereotypes, tagged values, and/or constraints into a package. Profiles have been used to create modeling extensions that can address specific types of implementation platforms, such as .NET, or specific modeling domains, such as embedded systems. Profiles are simply a convenience. In this text, we do not further describe the use of profiles.

OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN

Object-oriented approaches to developing information systems, technically speaking, can use any of the traditional methodologies presented in Chapter 1 (waterfall development, parallel development, phased development, prototyping, and throwaway prototyping). However, the object-oriented approaches are most associated with a *phased development RAD* methodology. The primary difference between a traditional approach like structured design and an object-oriented approach is how a problem is decomposed. In traditional approaches, the problem decomposition process is either process-centric or data-centric. However, when modeling real-world systems, processes and data are so closely related that it is difficult to pick one or the other as the primary focus. Based on this lack of congruence

⁵ For more specifics on the OCL, see Jos Warmer and Anneke Kleppe, *The Object Constraint Language: Precise Modeling with UML* (Reading, MA: Addison-Wesley, 1999) and the *UML 2.0 OCL Specification*, ptc/03-10-14 (www.uml.org <<http://www.uml.org/>>).

with the real-world, new *object-oriented methodologies* have emerged that use the RAD-based sequence of SDLC phases but attempt to balance the emphasis between process and data by focusing the decomposition of problems on objects that contain both data and processes. Both approaches are valid approaches to developing information systems. In this book, we focus only on object-oriented approaches.⁶

According to the creators of UML, Grady Booch, Ivar Jacobson, and James Rumbaugh,⁷ any modern object-oriented approach to developing information systems must be (1) use-case driven, (2) architecture-centric, and (3) iterative and incremental.

Use-Case Driven

Use-case driven means that *use cases* are the primary modeling tool to define the behavior of the system. A use case describes how the user interacts with the system to perform some activity, such as placing an order, making a reservation, or searching for information. The use cases are used to identify and to communicate the requirements for the system to the programmers who must write the system.

Use cases are inherently simple because they focus on only one activity at a time. In contrast, the process model diagrams used by traditional structured and RAD methodologies are far more complex because they require the system analyst and user to develop models of the entire system. With traditional methodologies, each business activity is decomposed into a set of subprocesses, which are, in turn, decomposed into further subprocesses, and so on. This goes on until no further process decomposition makes sense, and it often requires dozens of pages of interlocking diagrams. In contrast, use cases focus on only one activity at a time, so developing models is much simpler.⁸ We describe use cases and use case diagrams in Chapter 6.

Architecture Centric

Any modern approach to systems analysis and design should be architecture centric. *Architecture centric* means that the underlying software architecture of the evolving system specification drives the specification, construction, and documentation of the system. Modern object-oriented systems analysis and design approaches should support at least three separate but interrelated architectural views of a system: functional, static, and dynamic.

The *functional view* describes the external behavior of the system from the perspective of the user. Use cases and use case diagrams are the primary approach used to depict the functional view. Also, in some cases, activity diagrams are used to supplement use cases. The *static view* describes the structure of the system in terms of attributes, methods, classes, and relationships. The structure diagrams portray the static view of an evolving object-oriented information system. The *dynamic view* describes the internal behavior of the system in terms of messages passed among objects and state changes within an object. The dynamic view is represented in UML by behavior diagrams.

⁶ See Alan Dennis and Barbara Haley Wixom, *Systems Analysis and Design: An Applied Approach*, 2nd Ed. (New York: Wiley, 2003) for a description of the traditional approaches.

⁷ Grady Booch, Ivar Jacobson, and James Rumbaugh, *The Unified Modeling Language User Guide* (Reading, MA: Addison-Wesley, 1999).

⁸ For those of you that have experience with traditional structured analysis and design, this will be one of the most unusual aspects of object-oriented analysis and design using UML. Structured approaches emphasize the decomposition of the complete business process into subprocesses and sub-subprocesses. Object-oriented approaches stress focusing on just one use case activity at a time and distributing that single use case over a set of communicating and collaborating objects. Therefore, use case modeling may seem initially unsettling or counter-intuitive, but in the long run this single focus does make analysis and design simpler.

Iterative and Incremental

Modern object-oriented systems analysis and design approaches emphasize *iterative* and *incremental* development that undergoes continuous testing and refinement throughout the life of the project. Each iteration of the system brings it closer and closer to real user needs.

The Unified Process

The Unified Process is a specific methodology that maps out when and how to use the various UML techniques for object-oriented analysis and design. The primary contributors were Grady Booch, Ivar Jacobson, and James Rumbaugh of Rational. Whereas the UML provides structural support for developing the structure and behavior of an information system, the Unified Process provides the behavioral support. The Unified Process, of course, is use-case driven, architecture centric, and iterative and incremental.

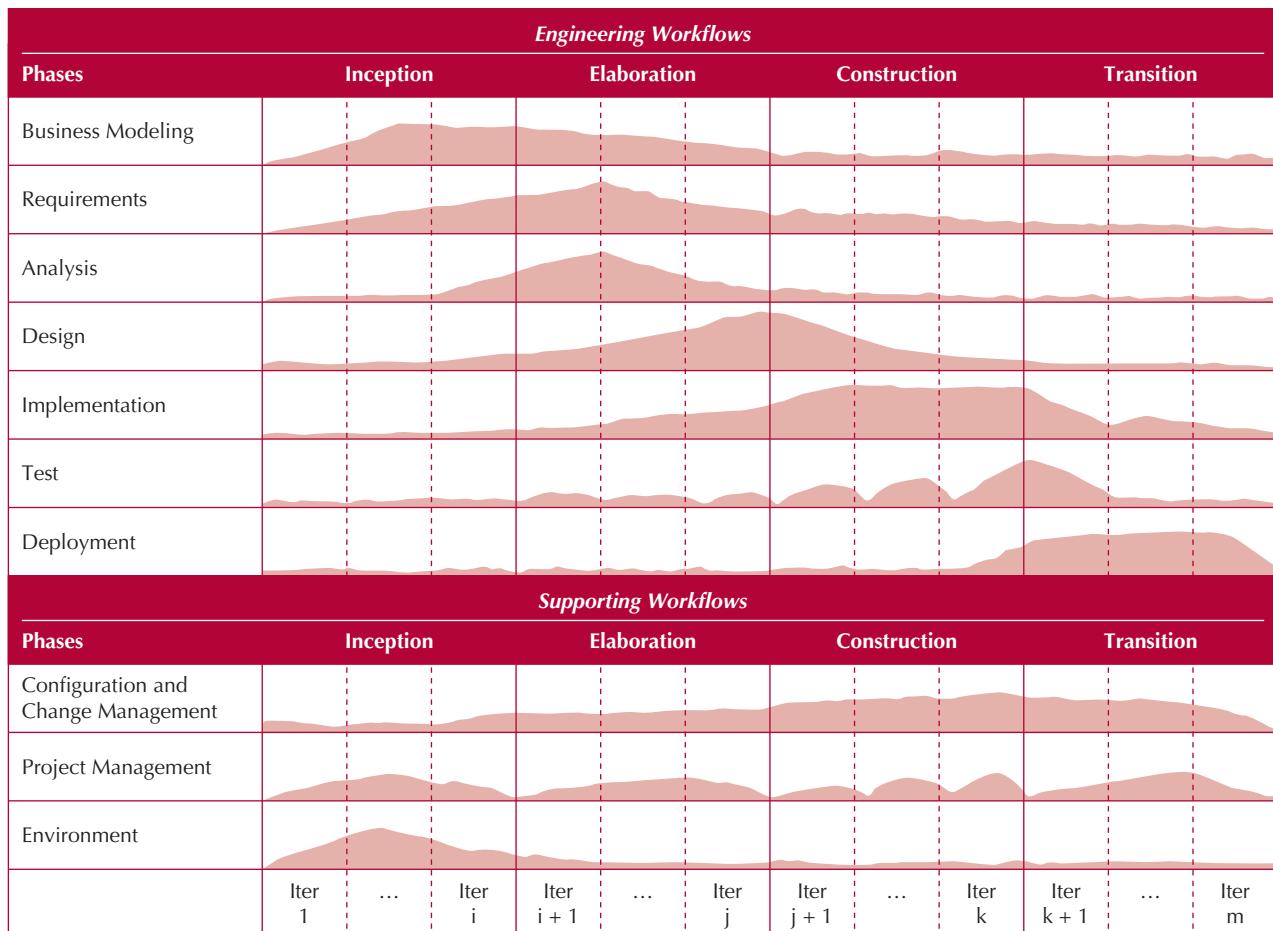
The Unified Process is a two-dimensional systems development process described by a set of phases and workflows. The phases are inception, elaboration, construction, and transition. The workflows include business modeling, requirements, analysis, design, implementation, test, deployment, project management, configuration and change management, and environment. In the remainder of this section, we describe the phases and workflows of the Unified Process.⁹ Figure 2-7 depicts the Unified Process.

Phases The *phases* of the Unified Process support an analyst in developing information systems in an iterative and incremental manner. The phases describe how an information system evolves through time. Depending on which development phase the evolving system is currently in, the level of activity will vary over the workflows. The curves, in Figure 2-7, associated with each workflow approximates the amount of activity that takes place during the specific phase. For example, the inception phase primarily involves the business modeling and requirements workflows, while practically ignoring the test and deployment workflows. Each phase contains a set of iterations, and each iteration uses the various workflows to create an incremental version of the evolving information system. As the system evolves through the phases, it improves and becomes more complete. Each phase has objectives, a focus of activity over the workflows, and incremental deliverables. Each of the phases is described as follows.

Inception In many ways, the *inception phase* is very similar to the planning phase of a traditional SDLC approach. In this phase, a business case is made for the proposed system. This includes feasibility analysis that should answer questions such as the following:

- Do we have the technical capability to build it? (technical feasibility)
- If we build it, will it provide business value? (economic feasibility)
- If we build it, will it be used by the organization? (organizational feasibility)

⁹ The material in this section is based on Khawar Zaman Ahmed and Cary E. Umrysh, *Developing Enterprise Java Applications with J2EE and UML* (Boston, MA: Addison-Wesley, 2002); Jim Arlow and Ilia Neustadt, *UML and The Unified Process: Practical Object-Oriented Analysis & Design* (Boston, MA: Addison-Wesley, 2002); Peter Eeles, Kelli Houston, Wojtek Kozaczynski, *Building J2EE Applications with the Rational Unified Process*, (Boston, MA: Addison-Wesley, 2003); Ivar Jacobson, Grady Booch, and James Rumbaugh, *The Unified Software Development Process* (Reading, MA: Addison-Wesley, 1999); Phillip Krutchen, *The Rational Unified Process: An Introduction, 2nd Ed.* (Boston, MA: Addison-Wesley, 2000).

**FIGURE 2-7** The Unified Process

To answer these questions, the development team performs work related primarily to the business modeling, requirements, and analysis workflows. In some cases, depending on the technical difficulties that could be encountered during the development of the system, a throw-away prototype is developed. This implies that the design, implementation, and test workflows also could be involved. The project management and environment supporting workflows are very relevant to this phase. The primary deliverables from the inception phase are: (1) a vision document that sets the scope of the project, identifies the primary requirements and constraints, sets up an initial project plan, and describes the feasibility of and risks associated with the project, and (2) the adoption of the necessary environment to develop the system.

Elaboration When one typically thinks about object-oriented systems analysis and design, the activities related to the *elaboration phase* of the Unified Process are the most relevant. The analysis and design workflows are the primary focus during this phase. The elaboration phase continues with developing the vision document, including finalizing the business case, revising the risk assessment, and completing a project plan in sufficient detail to allow the stakeholders to be able to agree with constructing the actual final system. It deals with gathering the requirements, building the UML structural and behavioral

models of the problem domain, and detailing the how the problem domain models fit into the evolving system architecture. Developers are involved with all but the deployment engineering workflow in this phase. As the developers iterate over the workflows, the importance of addressing configuration and change management becomes apparent. Also, the development tools acquired during the inception phase become critical to the success of the project during this phase.¹⁰ The primary deliverables of this phase include (1) the UML structure and behavior diagrams and (2) an executable of a baseline version of the evolving information system. The baseline version serves as the foundation for all later iterations. By providing a solid foundation at this point in time, the developers can begin to grow the system toward its completion in the construction and transition phases.

Construction The *construction phase*, as expected by its name, is heavily focused on programming the evolving information system. As such, it is primarily concerned with the implementation workflow. However, the requirements, analysis, and design workflows also are involved with this phase. It is during this phase that missing requirements are uncovered, and the analysis and design models are finally completed. Typically, there are iterations of the workflows during this phase, and during the last iteration, the deployment workflow kicks into high gear. The configuration and change management workflow, with its version control activities, becomes extremely important during the construction phase. At times, an iteration may have to be rolled back. Without good version controls, rolling back to a previous version (incremental implementation) of the system is nearly impossible. The primary deliverable of this phase is an implementation of the system that can be released for beta and acceptance testing.

Transition Like the construction phase, the *transition phase* addresses aspects typically associated with the implementation phase of a traditional SDLC approach. Its primary focus is on the testing and deployment workflows. Essentially, the business modeling, requirements, and analysis workflows should have been completed in earlier iterations of the evolving information system. Depending on the results from the testing workflow, it is possible that some redesign and programming activities on the design and implementation workflows could be necessary, but they should be minimal at this point in time. From a managerial perspective, the project management, configuration and change management, and environment are involved. Some of the activities that take place are beta and acceptance testing, fine tuning the design and implementation, user training, and the actual rolling out of the final product onto a production platform. Obviously, the primary deliverable is the actual executable information system. The other deliverables include user manuals, a plan to support the users, and a plan for upgrading the information system in the future.

Workflows The *workflows* describe the tasks or activities that a developer performs to evolve an information system over time. The workflows of the Unified Process are grouped into two broad categories: engineering and supporting. We describe each of the workflows as follows.

Engineering Workflows The *engineering workflows* include business modeling, requirements, analysis, design, implementation, test, and deployment workflows. The engineering workflows deal with the activities that produce the technical product (i.e., the information system). Next, we describe each engineering workflow.

¹⁰ With UML being comprised of fourteen different, related diagramming techniques, keeping the diagrams coordinated and the different versions of the evolving system synchronized is typically beyond the capabilities of a mere mortal systems developer. These tools typically include project management and CASE (Computer Aided Software Engineering) tools. We describe the use of these tools in Chapter 3.

Business Modeling. The *Business modeling* workflow uncovers problems and identifies potential projects within a user organization. This workflow aids management in understanding the scope of the projects that can improve the efficiency and effectiveness of a user organization. The primary purpose of business modeling is to ensure that both developer and user organizations understand where and how the to-be-developed information system fits into the business processes of the user organization. This workflow primarily is executed during the inception phase to ensure that we develop information systems that make business sense. The activities that take place on this workflow are most closely associated with the planning phase of the traditional SDLC; however, requirements gathering and use case and business process modeling techniques also are used to understand the business situation.

Requirements. In the Unified Process, the *requirements workflow* includes eliciting both functional and nonfunctional requirements. Typically, requirements are gathered from project stakeholders, such as end users, managers within the end user organization, and even customers. There are many different ways in which to capture requirements, including interviews, observation techniques, joint application development, document analysis, and questionnaires (see Chapter 5). As you should expect, the requirements workflow is utilized the most during the inception and elaboration phases. The identified requirements are very useful in developing the vision document and the use cases used throughout the development process. It should be stressed that additional requirements tend to be discovered throughout the development process. In fact, only the transition phase tends to have few if any additional requirements identified.

Analysis. The *analysis workflow* predominantly addresses creating an *analysis model* of the problem domain. In the Unified Process, the analyst begins designing the architecture associated with the problem domain, and using the UML, the analyst creates structural and behavioral diagrams that depict a description of the problem domain classes and their interactions. The primary purpose of the analysis workflow is to ensure that both the developer and user organizations understand the underlying problem and its domain without overanalyzing. If they are not careful, analysts can create *analysis paralysis*, which occurs when the project becomes so bogged down with analysis that the system is never actually designed or implemented. A second purpose of the analysis workflow is to identify useful reusable classes for class libraries. By reusing predefined classes, the analyst can avoid “reinventing the wheel” when creating the structural and behavioral diagrams. The analysis workflow is predominantly associated with the elaboration phase, but like the requirements workflow, it is possible that additional analysis will be required throughout the development process.

- *Design.* The *design workflow* transitions the analysis model into a form that can be used to implement the system: the *design model*. Where the analysis workflow concentrated on understanding the problem domain, the design workflow, focuses on developing a solution that will execute in a specific environment. Basically, the design workflow simply enhances the evolving information system description by adding classes that address the environment of the information system to the evolving analysis model. As such, the design workflow addresses activities, such as user interface design, database design, physical architecture design, detailed problem domain class design, and the optimization of the evolving information system. The design workflow primarily is associated with the elaboration and construction phases of the Unified Process.

- *Implementation.* The primary purpose of the *implementation workflow* is to create an executable solution based on the design model (i.e., programming). This includes not only writing new classes, but also incorporating reusable classes from executable class libraries into the evolving solution. As with any programming activity, testing of the new

classes and their interactions with the incorporated reusable classes must occur. Finally, in the case of multiple groups performing the implementation of the information system, the implementers also must integrate the separate, individually tested, modules to create an executable version of the system. The implementation workflow primarily is associated with the elaboration and construction phases.

- *Test.* The primary purpose of the *test workflow* is to increase the quality of the evolving system. As such, testing goes beyond the simple unit testing associated with the implementation workflow. In this case, testing also includes testing the integration of all modules used to implement the system, user acceptance testing, and the actual alpha testing of the software. Practically speaking, testing should go on throughout the development of the system; testing of the analysis and design models are involved during the elaboration and construction phases, while implementation testing is performed primarily during the construction and, to some degree, transition phases. Basically, at the end of each iteration during the development of the information system, some type of test should be performed.

- *Deployment.* The *deployment workflow* is most associated with the transition phase of the Unified Process. The deployment workflow includes activities, such as software packaging, distribution, installation, and beta testing. When actually deploying the new information system into a user organization, the developers may have to convert the current data, interface the new software with the existing software, and provide end user training on the use of the new system.

Supporting Workflows The supporting workflows include the project management, configuration and change management, and the environment workflows. The supporting workflows focus on the managerial aspects of information system development.

- *Project management.* While the other workflows associated with the Unified Process technically are active during all four phases, the *project management workflow* is the only truly cross-phase workflow. The development process supports incremental and iterative development, so information systems tend to grow or evolve over time. At the end of each iteration, a new incremental version of the system is ready for delivery. The project management workflow is quite important due to the complexity of the two-dimensional development model of the Unified Process (workflows and phases). This workflow's activities include risk identification and management, scope management, estimating the time to complete each iteration and the entire project, estimating the cost of the individual iteration and the whole project, and tracking the progress being made toward the final version of the evolving information system.

- *Configuration and change management.* The primary purpose of the *configuration and change management workflow* is to keep track of the state of the evolving system. In a nutshell, the evolving information system comprises a set of artifacts that includes, for example, diagrams, source code, and executables. During the development process, these artifacts are modified. The amount of work, and hence dollars, that goes into the development of the artifacts is substantial. As such, the artifacts themselves should be handled as any expensive asset would be handled—access controls must be put into place to safeguard the artifacts from being stolen or destroyed. Furthermore, since the artifacts are modified on a regular, if not continuous, basis, good version control mechanisms should be established. Finally, a good deal of project management information needs to be captured (e.g., author, time, and location of each modification). The configuration and change management workflow is associated mostly with the construction and transition phases.

- *Environment.* During the development of an information system, the development team needs to use different tools and processes. The *environment workflow*

addresses these needs. For example, a computer aided software engineering tool that supports the development of an object-oriented information system via the UML could be required. Other tools necessary would include programming environments, project management tools, and configuration management tools. The environment workflow includes acquiring and installing these tools. Even though this workflow can be active during all of the phases of the Unified Process, it should primarily be involved with the inception phase.

A MINIMALIST APPROACH TO OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN WITH UML 2.0

The UML is an object-oriented modeling language used to describe information systems. It provides a common vocabulary of object-oriented terms and a set of diagramming techniques that are rich enough to model any systems development project from analysis through implementation. Although the UML defines a large set of diagramming techniques, this book focuses on a smaller set of the most commonly used techniques. It should be stressed that UML is nothing more than a notation. Although unlikely, it is possible to develop an information system using a traditional approach with UML. The UML does not dictate any formal approach to developing information systems, but its iterative nature is best-suited to RAD-based approaches such as phased development (see Figure 1-4). A popular RAD-based approach that uses the UML is the Unified Process.

Benefits of Object-Oriented Systems Analysis and Design

So far we have described several major concepts that permeate the object-oriented approach, in general, and the UML 2.0 and Unified Process, in particular, but you may be wondering how these concepts affect the performance of a project team. The answer is simple. Concepts in the object-oriented approach enable analysts to break a complex system into smaller, more manageable modules, work on the modules individually, and easily piece the modules back together to form an information system. This modularity makes system development easier to grasp, easier to share among members of a project team, and easier to communicate to users who are needed to provide requirements and confirm how well the system meets the requirements throughout the SDLC.

By modularizing system development, the project team actually is creating reusable pieces that can be plugged into other systems efforts, or used as starting points for other projects. Ultimately, this can save time because new projects don't have to start completely from scratch.

Finally, many people argue that "object-think" is a much more realistic way to think about the real world. Users typically do not think in terms of data or process; instead, they see their business as a collection of logical units that contain both—so communicating in terms of objects improves the interaction between the user and the analyst or developer. Figure 2-8 summarizes the major concepts of the object-oriented approach and how each concept contributes to the benefits.

Extensions to the Unified Process

As large and as complex as the Unified Process is, many authors have pointed out a set of critical weaknesses. First, the Unified Process does not address staffing, budgeting, or contract management issues. These activities were explicitly left out of the Unified Process. Second, the Unified Process does not address issues relating to maintenance,

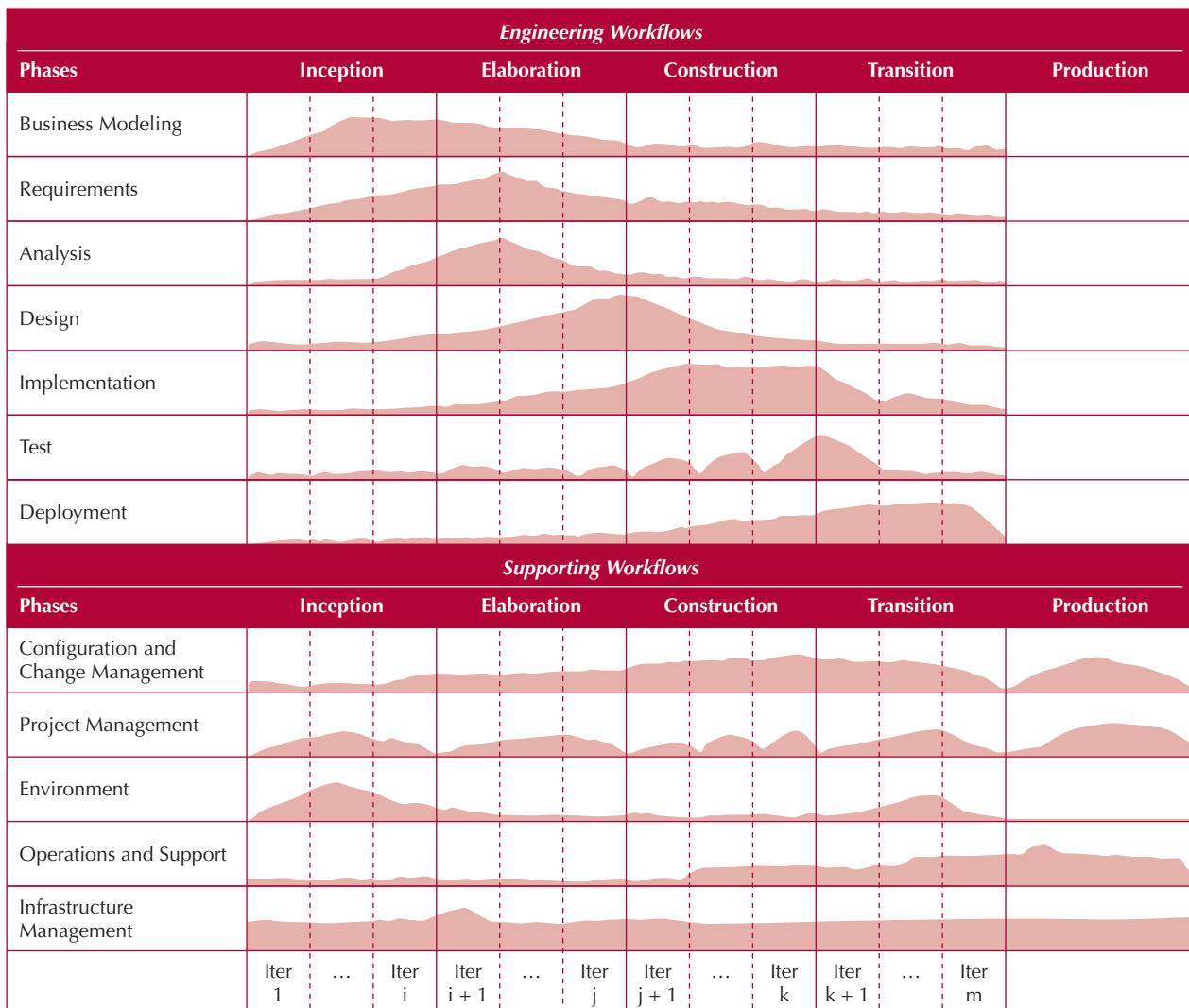
Concept	Supports	Leads to
Classes, objects, methods, and messages	<ul style="list-style-type: none"> ■ A more realistic way for people to think about their business ■ Highly cohesive units that contain both data and processes 	<ul style="list-style-type: none"> ■ Better communication between user and analyst or developer ■ Reusable objects ■ Benefits from having a highly cohesive system (see cohesion in Chapter 13)
Encapsulation and information hiding	<ul style="list-style-type: none"> ■ Loosely coupled units 	<ul style="list-style-type: none"> ■ Reusable objects ■ Fewer ripple effects from changes within an object or in the system itself ■ Benefits from having a loosely coupled system design (see coupling in Chapter 13)
Inheritance	<ul style="list-style-type: none"> ■ Allows us to use classes as standard templates from which other classes can be built 	<ul style="list-style-type: none"> ■ Less redundancy ■ Faster creation of new classes ■ Standards and consistency within and across development efforts ■ Ease in supporting exceptions
Polymorphism and Dynamic Binding	<ul style="list-style-type: none"> ■ Minimal messaging that is interpreted by objects themselves 	<ul style="list-style-type: none"> ■ Simpler programming of events ■ Ease in replacing or changing objects in a system ■ Fewer ripple effects from changes within an object or in the system itself
Use-case driven and use cases	<ul style="list-style-type: none"> ■ Allows users and analysts to focus on how a user will interact with the system to perform a single activity 	<ul style="list-style-type: none"> ■ Better understanding and gathering of user needs ■ Better communication between user and analyst
Architecture centric and functional, static, and dynamic views	<ul style="list-style-type: none"> ■ Viewing the evolving system from multiple points of view 	<ul style="list-style-type: none"> ■ Better understanding and modeling of user needs ■ More complete depiction of information system
Iterative and incremental development	<ul style="list-style-type: none"> ■ Continuous testing and refinement of the evolving system 	<ul style="list-style-type: none"> ■ Meeting real needs of users ■ Higher quality systems

FIGURE 2-8 Benefits of the Object Approach

operations, or support of the product once it has been delivered. As such, it is not a complete software process; it is only a development process. Third, the Unified Process does not address cross- or inter-project issues. Considering the importance of reuse in object-oriented systems development and the fact that in many organizations employees work on many different projects at the same time, leaving out inter-project issues is a major omission.

To address these omissions, Ambler and Constantine suggest the addition of a Production phase and two workflows: the Operations and Support workflow and the Infrastructure Management workflow (see Figure 2-9).¹¹ In addition to these new workflows, the test, deployment and environment workflows are modified, and the project management and configuration and change management workflows are extended into the production phase. These extensions are based on alternative object-oriented software

¹¹ S.W. Ambler and L.L. Constantine, *The Unified Process Inception Phase: Best Practices in Implementing the U* (CMP Books, 2000); S.W. Ambler and L.L. Constantine, *The Unified Process Elaboration Phase: Best Practices in Implementing the UP* (CMP Books, 2000); S.W. Ambler and L.L. Constantine, *The Unified Process Construction Phase: Best Practices in Implementing the UP* (CMP Books, 2000); S.W. Ambler and L.L. Constantine, *The Unified Process Transition and Production Phases: Best Practices in Implementing the UP* (CMP Books, 2002).

**FIGURE 2-9** The Enhanced Unified Process

processes: the OPEN process and the Object-Oriented Software Process.¹² The new phase, new workflows, and the modifications and extensions to the existing workflows are described next.

Production Phase The *production phase* is concerned primarily with issues related to the software product after it has been successfully deployed. As you should expect, the phase focuses on issues related to updating, maintaining, and operating the software. Unlike the previous phases, there are no iterations or incremental deliverables. If

¹² S.W. Ambler, *Process Patterns—Building Large-Scale Systems Using Object Technology* (Cambridge, UK: SIGS Books/Cambridge University Press, 1998); S.W. Ambler, *More Process Patterns—Delivering Large-Scale Systems Using Object Technology* (Cambridge, UK: SIGS Books/Cambridge University Press, 1999); I. Graham, B. Henderson-Sellers, and H. Younessi, *The OPEN Process Specification* (Harlow, UK: Addison-Wesley, 1997); B. Henderson-Sellers and B. Unhelkar, *OPEN modeling with UML* (Harlow, UK: Addison-Wesley, 2000).

a new release of the software is to be developed, then the developers must begin a new run through the first four phases again. Based on the activities that take place during this phase, no engineering workflows are relevant. The supporting workflows that are active during this phase include the configuration and change management workflow, the project management workflow, the new operations and support workflow, and the infrastructure management workflow.

Operations and Support Workflow The *operations and support workflow*, as you might guess, addresses issues related to supporting the current version of the software and operating the software on a daily basis. Activities include creating plans for the operation and support of the software product once it has been deployed, creating training and user documentation, putting into place necessary backup procedures, monitoring and optimizing the performance of the software, and performing corrective maintenance on the software. This workflow becomes active during the construction phase and increases in level of activity throughout the transition and finally, the production phase. The workflow finally drops off when the current version of the software is replaced by a new version. Many developers are under the false impression that once the software has been delivered to the customer, their work is finished. In most cases, the work of supporting the software product is much more costly and time consuming than the original development. As such, as a developer, your work may have just begun.

Infrastructure Management Workflow The *infrastructure management workflow's* primary purpose is to support the development of the infrastructure necessary to develop object-oriented systems. Activities like development and modification of libraries, standards, and enterprise models are very important. When the development and maintenance of a problem domain architecture model goes beyond the scope of a single project and reuse is going to occur, the infrastructure management workflow is essential. Another very important set of cross-project activities is the improvement of the software development process. Since the activities on this workflow tend to affect many projects and the Unified Process only focuses on a specific project, the Unified Process tends to ignore these activities (i.e., they are simply beyond the scope and purpose of the Unified Process).

Existing Workflow Modifications and Extensions In addition to the workflows that were added to address deficiencies contained in the Unified Process, existing workflows had to be modified and/or extended into the production phase. These workflows include the test, deployment, environment, project management, and configuration and change management workflows. Each of the enhancements is described next.

Test For information systems of high quality to be developed, testing should be done on every deliverable, including those created during the inception phase. Otherwise, less than quality systems will be delivered to the customer.

Deployment In most corporations today, legacy systems exist, and these systems have databases associated with them that must be converted to interact with the new systems. Due to the complexity of deploying new systems, the conversion requires significant planning. As such, the activities on the deployment workflow need to begin in the inception phase instead of waiting until the end of the construction phase as suggested by the Unified Process.

Environment The environment workflow needed to be modified to include activities related to setting up the operations and production environment. The actual work performed is similar to the work related to setting up the development environment that was performed during the inception phase. In this case, the additional work is performed during the transition phase.

Project Management Even though this workflow does not include staffing the project, managing the contracts among the customers and vendors, and managing the project's budget, these activities are crucial to the success of any software development project. As such, we suggest extending project management to include these activities. Furthermore, this workflow should additionally occur in the production phase to address issues such as training, staff management, and client relationship management.

Configuration and Change Management The configuration and change management workflow is extended into the new production phase. Activities performed during the production phase include identifying potential improvements to the operational system and assessing the potential impact of the proposed changes. Once these changes have been identified and their impact understood, the developers can schedule the changes to be made and deployed with future releases.

The Minimalist Object-Oriented Systems Analysis and Design Approach

As we stated previously, object-oriented systems analysis and design (OOSAD) approaches are based on a phased-development RAD approach. However, because of the iteration across the functional, static, and dynamic views of the evolving information system, an actual object-oriented development process tends to be more complex than typical phased-development RAD approaches. For example, the two-dimensional model of the Unified Process and the identified extensions is much more complex than a typical phased-development RAD approach (compare Figures 1-4 and 2-9). From a learning perspective, the complexity of the enhanced Unified Process makes putting it into practice in the classroom practically impossible. As such, we have followed a minimalist style¹³ of presenting a generic approach to OOSAD. The minimalist OOSAD (MOOSAD) approach that we present in this section is based on the Unified Process as extended by the processes associated with the OPEN Process and the Object-Oriented Software Process approaches to object-oriented systems development. We also have included concepts from XP¹⁴ to help in controlling the complexity of the development process. Finally, due to the size and complexity of the UML, we only use a minimal set of the UML with our minimalist approach.¹⁵

Figure 2-10 portrays our modified phased-development RAD-based approach. The solid lines in Figure 2-10 represent information flows from one step in our approach to

¹³ John M. Carroll, *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill* (Cambridge, MA: MIT Press, 1990).

¹⁴ For more information, see K. Beck, *eXtreme Programming Explained: Embrace Change* (Reading, MA: Addison-Wesley, 2000), M. Lippert, S. Rock, and H. Wolf, *eXtreme Programming in Action: Practical Experiences from Real World Projects* (New York: Wiley, 2002), or online at www.extremeprogramming.com. Also, see discussion in Chapter 1.

¹⁵ In many places, the UML is not sufficient for our purposes. For example, the UML does not have any useful diagrams to design user interfaces. In cases where the UML is not sufficient, we will use extensions. However, our overall intention is to minimize both the size and complexity of the UML for learning purposes. For more information on the UML see www.uml.org.

another step. The dashed lines represent feedback from a later step to an earlier one. For example, plans flow from the planning step into the requirements determination and use case development step, and feedback from the requirements determination and use case development step flows back to the planning step.

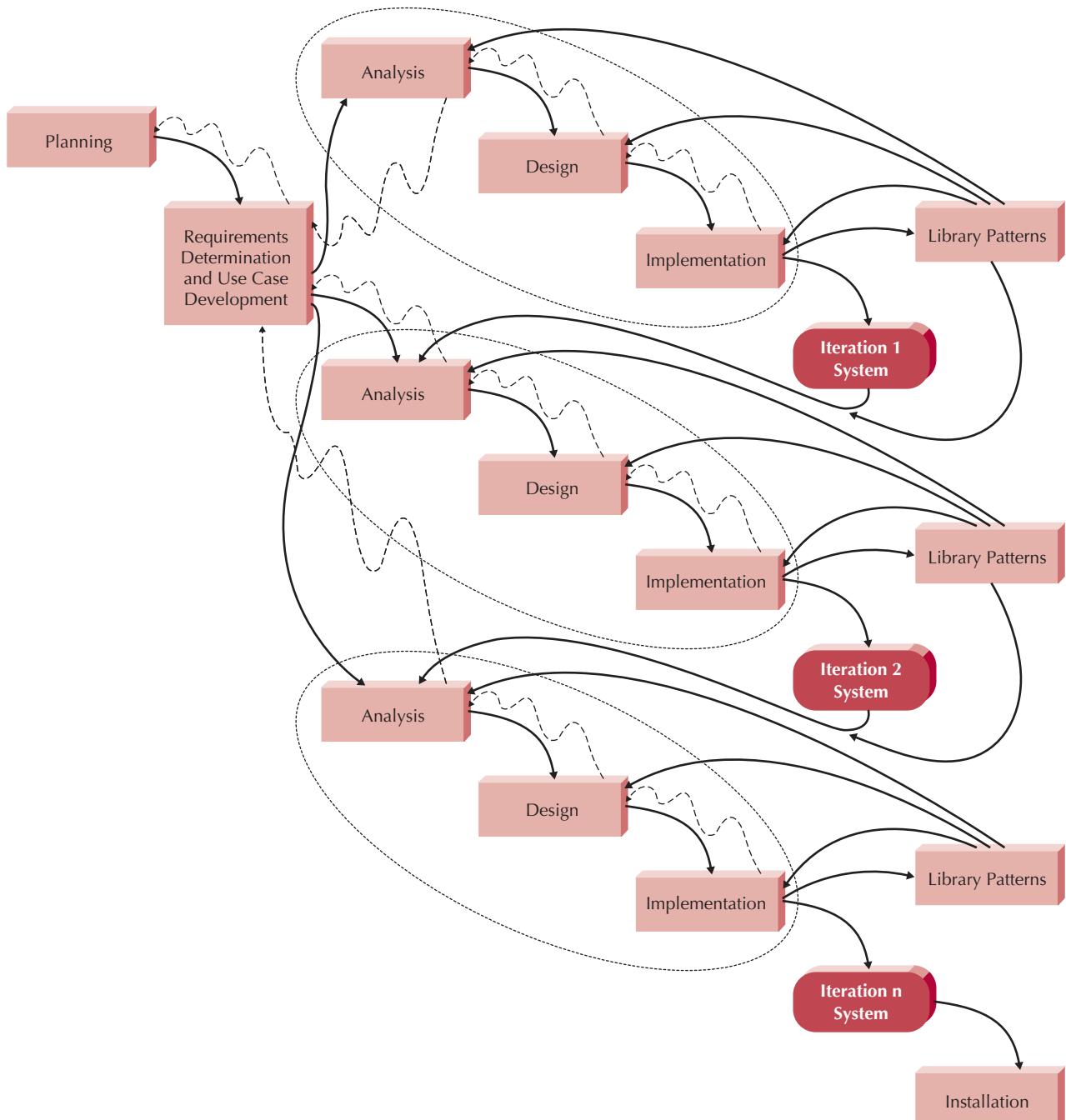


FIGURE 2-10 The Minimalist Object-Oriented Systems Analysis and Design (MOOSAD) Approach

The first step of the approach is *planning*. If you will recall, planning deals with understanding *why* an information system should be built, and determining how the project team will go about building it. The second step of the approach is *requirements determination* and *use case development*. As mentioned above, use cases identify and communicate the high-level requirements for the system (i.e., they provide a functional or external behavioral view of the system). Use cases and the use case model drive the remaining steps in our approach (i.e., all of the information required from the remaining steps in our approach is derived from the use cases and the use case model).

Next, the developers of the system perform a *build*. Each build makes incremental progress toward delivering the entire system. The first build is based on the use cases with the highest priority. Builds are performed until the system is complete. Each build comprises an analysis, design, and implementation step. Each build provides feedback to the requirements determination and use case development step and delivers a functional system to the next build and a set of *patterns* that can be included in the library.¹⁶ Later builds incorporate additional lower-priority and newly identified use cases. Again, each build is based on the remaining use cases with the highest priority. For project management purposes, a build utilizes the idea of *timeboxing*.¹⁷ Typically, in object-oriented systems development, the time frame for each timebox varies from one to two weeks to one to two months, depending on the size and complexity of the project.

Figure 2-11 maps the enhanced Unified Process's phases and workflows into our minimalist approach and the relevant chapters in which the material is covered. We use our minimalist OOSAD approach in this textbook only to simplify the learning experience. You should realize that a two-dimensional model of the development process is more realistic in practice.

As Figure 2-11 shows, in this text we are focused primarily in the area associated with the elaboration phase and the requirements, analysis, design, and project management workflows of the extended Unified Process. In many object-oriented systems development environments today, code generation is supported. Thus, from a business perspective, we believe the activities associated with these workflows are the most important. The remainder of the book is organized around the steps in our MOOSAD approach (see Figure 2-12).

SUMMARY

Today, the most exciting change to systems analysis and design is the move to object-oriented techniques, which view a system as a collection of self-contained objects that have both data and processes. However, the ideas underlying object-oriented techniques are simply ideas that have either evolved from traditional approaches to systems development or they are old ideas that have now become practical due to the cost-performance ratios of modern computer hardware in comparison to the ratios of the past. Today, the cost of developing modern software is composed primarily of the cost associated with the developers themselves and not the computers. As such, object-oriented approaches to developing information systems hold out much promise in controlling these costs.

¹⁶ A pattern is a useful group of collaborating classes that provide a solution to a commonly occurring problem. We describe the use of patterns in more detail in Chapter 7.

¹⁷ Timeboxing sets a fixed deadline for a project and delivers the system by that deadline no matter what, even if the functionality needs to be reduced. Timeboxing ensures that project teams don't get hung up on the final "finishing touches" that can drag out indefinitely. Timeboxing is covered in more detail in Chapter 4.

Enhanced UP Phases	MOOSAD Steps	Chapters
Inception	Planning Requirements Determination & Use Case Development	3–6
Elaboration	Requirements Determination & Use Case Development Analysis Design	5–13
Construction	Implementation	11, 14
Transition	Implementation Installation	14, 15
Production	Installation	15

Enhanced UP Engineering Workflows	MOOSAD Steps	Chapters
Business Modeling	Planning	3, 5–7
Requirements	Requirements Determination & Use Case Development	5–7, 12
Analysis	Analysis	6–8
Design	Design	9–13
Implementation	Implementation	11, 14
Test	Implementation	14
Deployment	Installation	15

Enhanced UP Supporting Workflows	MOOSAD Steps	Chapters
Project Management	Across Steps	3, 4, 6, 15
Configuration and Change Management	Across Steps	3, 15
Environment	Across Steps	4
Operations and Support	Installation	15
Infrastructure Management	Across Steps	4

FIGURE 2-11
The Enhanced Unified
Process and the
MOOSAD Approach

Basic Characteristics of Object-Oriented Systems

An object is a person, place, or thing about which we want to capture information. Each object has attributes that describe information about it and behaviors, which are described by methods that specify what an object can do. Objects are grouped into classes, which are collections of objects that share common attributes and methods. The classes can be arranged in a hierarchical fashion in which low-level classes, or subclasses, inherit attributes and methods from superclasses above them to reduce the redundancy in development. Objects communicate with each other by sending messages, which trigger methods. Polymorphism allows a message to be interpreted differently by different kinds of objects. This form of communication allows us to treat objects like black boxes and ignore the inner workings of the objects. The idea of concealing an object's inner processes and data from the outside is known as encapsulation. The benefit of these object concepts is a modular, reusable development process.

Chapter	Step	Sample Techniques	Deliverable
3	Identifying Business Value	System Request	System Request
	Analyze Feasibility	Technical Feasibility, Economic Feasibility, Organizational Feasibility	Feasibility Study
4	Develop Workplan Time Estimation	Task Identification	Workplan
	Staff the Project Creating a Project Charter	Creating a Staffing Plan Project Charter	Staffing Plan
	Control and Direct Project Track Tasks Coordinate Project Manage Scope Mitigate Risk	Refine Estimates PERT/CPM CASE Tool Standards List Risk Assessment	GANTT Chart
5	Requirements Determination	Improvement Identification Techniques Interviews JAD Questionnaires	Information
6	Functional Modeling	Activity Diagram Use Cases Use Case Diagram Use Case Point Estimation	Functional Model Use Case Points
7	Structural Modeling	CRC Cards Class Diagram Object Diagram	Structural Models
8	Behavioral Modeling	Sequence Diagram Communication Diagram Behavioral State Machine	Dynamic Models
9	Moving on to Design	Factoring Partitions and Layers Package Diagrams Custom Development Package Development Outsourcing	Factored Models Design Strategy

FIGURE 2-12 Textbook Overview (*Continues*)

Unified Modeling Language

The Unified Modeling Language, or UML, is a standard set of diagramming techniques that provide a graphical representation that is rich enough to model any systems development project from analysis through implementation. Today most object-oriented systems analysis and design approaches use the UML to depict an evolving system. The UML uses a set of different diagrams to portray the various views of the evolving system. The diagrams are grouped into two broad classifications: structure and behavior. The structure diagrams include class, object, package, deployment, component, and composite structure diagrams. The behavior diagrams

Chapter	Step	Sample Techniques	Deliverable
10	Class and Method Design	Reuse Factoring Design Optimization Constraints Method Specification Activity Diagram	Restructured Models Class Design Contracts Method Design
11	Data Management Layer Design	Selecting an Object Persistence Format Mapping Problem Domain Objects to Object Persistence Formats Optimizing Object Persistence Designing Data Access and Manipulation Classes	Object Persistence Design Data Access and Manipulation Design
12	Human Computer Interaction Layer Design	Windows Navigation Diagram Real Use Cases Input Design Output Design	Interface Design
13	Physical Architecture Layer Design	Hardware Design System Software Design Network Design	Physical Architecture Design Infrastructure Design
14	Construction	Software Testing	Test Plan Documentation Completed System
15	Installation	Direct Conversion Parallel Conversion Phased Conversion	Conversion Plan Training Plan
	Operations and Support	Support Strategy Post-Implementation Review	Support Plan

FIGURE 2-12 Textbook Overview (*Continued*)

include activity, sequence, communication, interaction overview, timing, behavior state machine, protocol state machine, and use case diagrams.

Object-Oriented Systems Analysis and Design

Object-oriented systems analysis and design (OOSAD) are most associated with a phased development RAD-based methodology where the time spent in each phase is very short. OOSAD uses a use-case driven, architecture centric, iterative, and incremental information systems development approach. It supports three different views of the evolving system: functional, static, and dynamic. OOSAD allows the analyst to decompose complex problems into smaller, more manageable components using a commonly accepted set of notations. Also, many people believe that users do not think in terms of data or processes, but instead think in terms of a collection of collaborating objects. As such, object-oriented systems analysis and design allows the analyst to interact with the user using objects from the user's environment instead of a set of separate processes and data.

**YOUR
TURN**

2-6 OO Systems Analysis and Design Methodology

Review Figures 2-7, 2-9, 2-10, and 2-11. Based on your understanding of the UP, the EUP, and MOOSAD, suggest a set of steps for an alternative object-oriented

systems development method. Be sure that the steps will be capable of delivering an executable and maintainable system.

One of the most popular approaches to object-oriented systems analysis and design is the Unified Process. The Unified Process is a two-dimensional systems development process described with a set of phases and workflows. The phases consist of the inception, elaboration, construction, and transition phases. The workflows are organized into two subcategories: engineering and supporting. The engineering workflows include business modeling, requirements, analysis, design, implementation, test, and deployment workflows, while the supporting workflows comprise the project management, configuration and change management, and the environment workflows. Depending on which development phase the evolving system is currently in, the level of activity will vary over the workflows.

A Minimalist Approach to Object-Oriented Systems Analysis and Design with UML

The Minimalist OOSAD (MOOSAD) approach is based on the processes described in the Unified Process, the Open Process, the Object-Oriented Software Process, and XP approaches to object-oriented systems development. It uses a modified phased-delivery RAD approach to its life cycle. It is also use-case driven, architecture centric, and iterative and incremental. It supports three different generic views of the evolving system: functional, static, and dynamic. Furthermore, it utilizes timeboxes to manage the creation of builds of the system. Finally, it uses UML 2.0 as its graphical notation to support the structural and behavioral modeling of the system.

KEY TERMS

A-Kind-Of (AKO)	Concrete classes	Infrastructure management workflow
Abstract classes	Configuration and change management workflow	Implementation workflow
Activity diagram	Constraints	Information hiding
Analysis model	Construction phase	Inherit
Analysis paralysis	Deployment diagram	Inheritance
Analysis workflow	Deployment workflow	Instance
Architecture centric	Design model	Interaction diagram
Attribute	Design workflow	Interaction overview diagram
Behavior	Dynamic binding	Iterative
Behavior diagrams	Dynamic view	Message
Behavior state machine	Elaboration phase	Method
Build	Encapsulation	Object
Business modeling workflow	Engineering workflows	Object diagram
Class	Environment workflow	Object management group (OMG)
Class diagram	Functional view	Object-oriented approach
Communication diagram	Inception phase	Object-oriented methodologies
Component diagram	Incremental	Operations and support workflow
Composite structure diagram		Package

Package diagram	Simula	Timeboxing
Pattern	Smalltalk	Timing diagram
Phased development RAD	State	Transition phase
Phases	State machine	Unified Modeling Language (UML)
Planning	Static binding	Use case
Polymorphism	Static view	Use case development
Production phase	Stereotypes	Use case diagram
Profiles	Structure diagrams	Use-case driven
Project management workflow	Subclass	Workflows
Protocol state machine	Superclass	
Requirements determination	Supporting workflows	
Requirements workflow	Tagged Values	
Sequence diagram	Test workflow	

QUESTIONS

1. Describe the major elements and issues with an object-oriented approach to developing information systems.
2. What is the difference between classes and objects?
3. What are methods and messages?
4. Why are encapsulation and information hiding important characteristics of object-oriented systems?
5. What is meant by polymorphism when applied to object-oriented systems?
6. Compare and contrast dynamic and static binding.
7. What is the Unified Modeling Language?
8. Who is the Object Management Group?
9. What is the primary purpose of structure diagrams?
10. For what are behavior diagrams used?
11. What is a use case?
12. What is meant by use-case driven?
13. Why is it important for an OOSAD approach to be architecture centric?
14. What does it mean for an OOSAD approach to be incremental and iterative?
15. What are the phases and workflows of the Unified Process?
16. Compare the phases of the Unified Process with the phases of the waterfall model described in Chapter 1.
17. What are the benefits of an object-oriented approach to systems analysis and design?
18. Compare and contrast the typical phased delivery RAD SDLC with the modified-phased delivery RAD SDLC associated with the OOSAD approach described in this chapter.
19. What are the steps or phases of the minimalist OOSAD approach described in this chapter?
20. What are the different views supported by the minimalist OOSAD approach described in this chapter?
21. What diagrams and models support the different views identified in the previous question?
22. What is a build?
23. What is a pattern?
24. How do the Unified Process's phases and workflows map into the steps of the minimalist OOSAD approach described in this chapter?

EXERCISES

- A. Investigate the Unified Modeling Language on the Web. Write a paragraph news brief describing the current state of the UML. (*Hint:* A good Web site to begin with is www.uml.org.)
- B. Investigate Rational's Unified Process (RUP) on the Web. RUP is a commercial version that extends aspects of the Unified Process. Write a brief memo describing how it is related to the Unified Process as described in this chapter. (*Hint:* A good Web site to begin with is www.rational.com.)
- C. Investigate the Object Management Group (OMG) on the Web. Write a report describing the purpose of the OMG and what it is involved with, besides the UML. (*Hint:* A good Web site to begin with is www.omg.org.)
- D. Using the Web, find a set of CASE tools that support the UML. A couple of examples include Rational Rose and Poseidon. Find at least two additional ones. Write a short report describing how well they support the UML, and make a recommendation as to which one you believe would be best for a project team to use in developing an

- object-oriented information system using the UML.
- E. Suppose you are a project manager that typically has been using a waterfall development-based methodology on a large and complex project. Your manager has just read the latest article in Computerworld that advocates replacing this methodology with the Unified Process and comes to your office requesting you to switch. What do you say?
- F. Suppose you were an analyst working for a small company to develop an accounting system. Would you use the Unified Process to develop the system, or would you prefer one of the traditional approaches described in Chapter 1? Why?
- G. Suppose you were an analyst developing a new information system to automate the sales transactions and manage inventory for each retail store in a large chain. The system would be installed at each store and exchange data with a mainframe computer at the company's head office. Would you use the Unified Process to develop the system or would you prefer one of the traditional approaches described in Chapter 1? Why?

MINICASES

1. Joe Brown, the president of Roanoke Manufacturing, requested that Jack Jones, the MIS department manager, to investigate the viability of selling their products over the Web. Currently, the MIS department is still using an IBM mainframe as their primary deployment environment. As a first step, Jack contacted his friends at IBM to see if they had any suggestions as to how Roanoke Manufacturing could move toward supporting sales in an electronic commerce environment while keeping their mainframe as their main system. His friends explained that IBM (www.ibm.com) now supports Java and Lynix on their mainframes. Furthermore, Jack learned that IBM owns Rational (www.rational.com), the creator of the UML and the Unified Precess. As such, they suggested that Jack investigate using object-oriented systems as a basis for developing the new system. They also suggested that using the Rational Unified Process (RUP), Java, and virtual Lynix machines on his current mainframe as a way to support the movement toward a distributed electronic commerce system would protect his current investment in his legacy systems while allowing the new system to be developed in a more modern manner.

Even though Jack's IBM friends were very persuasive, Jack is still a little wary about moving his operation from a structured systems approach to this new object-oriented approach. Assuming that you are one of Jack's IBM friends, how would you convince him to move

toward using an object-oriented systems development method, such as RUP, and using Java and Lynix as a basis for developing and deploying the new system on Roanoke Manufacturing's current mainframe.

2. While recently attending a software development conference, Susan Brown, a systems analyst at Staunton Consulting, was exposed to object-oriented approaches to developing software. Based on what she learned, she feels that object-oriented approaches are a must for her firm to survive in the consulting business in the future. the advantages of using object-oriented approaches seem to vastly out weigh the benefits of remaining with the software development methods used by Staunton Consulting. However, even though she has 15 years of software development experience, she feels somewhat overwhelmed with the complexity of using UML 2.0, with its 14 diagrams, and the Enhanced Unified Process.

Assuming that you are a close friend that has been using your own firm's UML 1.3 based object-oriented systems development method for the past 5 years, how would you help Susan overcome her concern with the complexity of UML 2.0 and the Enhanced Unified Process? Furthermore, it would be helpful if you helped her create a short checklist of characteristics of software development projects that she could use to base her recommendation for her firm to switch to an object-oriented systems development approach that used UML 2.0.

PART ONE

PLANNING PHASE

The Planning Phase is the fundamental process of understanding why an information system should be built, and determining how the project team will build it. The deliverables are combined into a system request which is presented to the project sponsor and approval committee at the end of this phase. They decide whether it is advisable to proceed with the system.

CHAPTER 3

PROJECT INITIATION

This chapter describes Project Initiation, the point at which an organization creates and assesses the original goals and expectations for a new system. The first step in the process is to identify a project that will deliver value to the business and to create a system request that provides basic information about the proposed system. Next the analysts perform a feasibility analysis to determine the technical, economic, and organizational feasibility of the system, and if appropriate, the system is selected, and the development project begins.

OBJECTIVES

- Understand the importance of linking the information system to business needs.
- Be able to create a system request.
- Understand how to assess technical, economic, and organizational feasibility.
- Be able to perform a feasibility analysis.
- Understand how projects are selected in some organizations.

CHAPTER OUTLINE

Introduction	Economic Feasibility
Project Identification	Organizational Feasibility
System Request	Applying the Concepts at CD Selections
Applying the Concepts at CD Selections	Project Selection
Feasibility Analysis	Applying the Concepts at CD Selections
Technical Feasibility	Summary

INTRODUCTION

The first step in any new development project is for someone—a manager, staff member, sales representative, or systems analyst—to see an opportunity to improve the business. New systems start first and foremost from some business need or opportunity. Many ideas for new systems or improvements to existing ones arise from the application of a new technology, but an understanding of technology is usually secondary to a solid understanding of the business and its objectives.

This may sound like common sense, but unfortunately, many projects are started without a clear understanding of how the system will improve the business. The IS field is filled with thousands of buzzwords, fads, and trends (e.g., customer relationship management [CRM], mobile computing, data mining). The promise of these innovations can appear so attractive that organizations begin projects even if they are not sure what value they offer,

because they believe that the technologies are somehow important in their own right. A 1996 survey by the Standish Group found that 42 percent of all corporate IS projects were abandoned before completion; a similar 1996 study by the General Accounting Office found 53 percent of all U.S. government IS projects were abandoned. Most times, problems can be traced back to the very beginning of the SDLC where too little attention was given to the identifying business value and understanding the risks associated with the project.

Does this mean that technical people should not recommend new systems projects? Absolutely not. In fact, the ideal situation is for both IT people (i.e., the experts in systems) and the business people (i.e., the experts in business) to work closely to find ways for technology to support business needs. In this way, organizations can leverage the exciting technologies that are available while ensuring that projects are based upon real business objectives, such as increasing sales, improving customer service, and decreasing operating expenses. Ultimately, information systems need to affect the organization's bottom line (in a positive way!).

In general, a *project* is a set of activities with a starting point and an ending point meant to create a system that brings value to the business. *Project initiation* begins when someone (or some group) in the organization (called the *project sponsor*) identifies some business value that can be gained from using information technology. The proposed project is described briefly using a technique called the system request, which is submitted to an approval committee for consideration. The approval committee reviews the system request and makes an initial determination, based on the information provided, of whether to investigate the proposal or not. If so, the next step is the feasibility analysis.

The feasibility analysis plays an important role in deciding whether to proceed with an IS development project. It examines the technical, economic, and organizational pros and cons of developing the system, and it gives the organization a slightly more detailed picture of the advantages of investing in the system as well as any obstacles that could arise. In most cases, the project sponsor works together with an analyst (or analyst team) to develop the feasibility analysis for the approval committee.

Once the feasibility analysis has been completed, it is submitted back to the approval committee along with a revised system request. The committee then decides whether to approve the project, decline the project, or table it until additional information is available. Projects are selected by weighing risks and return, and by making trade-offs at the organizational level.

CONCEPTS

IN ACTION

3-A Interview with Lyn McDermid, CIO, Dominion Virginia Power

A CIO needs to have a global view when identifying and selecting projects for her organization. I would get lost in the trees if I were to manage on a project-by-project basis. Given this, I categorize my projects according to my three roles as a CIO, and the mix of my project portfolio changes depending on the current business environment.

My primary role is to **keep the business running**. That means every day when each person comes to work, they can perform his or her job efficiently. I measure this using various service level, cost, and productivity measures. Projects that keep the business running could have a high priority if the business were in the middle of a merger, or a low priority if things were running smoothly, and it were "business as usual."

My second role is to push **innovation that creates value for the business**. I manage this by looking at our lines of business and asking which lines of business create the most value for the company. These are the areas for which I should be providing the most value. For example, if we had a highly innovative marketing strategy, I would push for innovation there. If operations were running smoothly, I would push less for innovation in that area.

My third role is strategic, to look beyond today and find **new opportunities** for both IT and the business of providing energy. This may include investigating process systems, such as automated meter reading or looking into the possibilities of wireless technologies.

—Lyn McDermid

PROJECT IDENTIFICATION

A project is identified when someone in the organization identifies a *business need* to build a system. This could occur within a business unit or IT, by a steering committee charged with identifying business opportunities, or evolve from a recommendation made by external consultants. Examples of business needs include supporting a new marketing campaign, reaching out to a new type of customer, or improving interactions with suppliers. Sometimes, needs arise from some kind of “pain” within the organization, such as a drop in market share, poor customer service levels, or increased competition. Other times, new business initiatives and strategies are created, and a system is required to enable them.

Business needs also can surface when the organization identifies unique and competitive ways of using IT. Many organizations keep an eye on *emerging technology*, which is technology that is still being developed and not yet viable for widespread business use. For example, if companies stay abreast of technology like the Internet, smart cards, and scent technology in their earliest stages, they can develop business strategies that leverage the capabilities of these technologies and introduce them into the marketplace as a *first mover*. Ideally, they can take advantage of this first-mover advantage by making money and continuing to innovate while competitors trail behind.

The *project sponsor* is someone who recognizes the strong business need for a system and has an interest in seeing the system succeed. He or she will work throughout the SDLC to make sure that the project is moving in the right direction from the perspective of the business. The project sponsor serves as the primary point of contact for the system. Usually the sponsor of the project is from a business function, such as Marketing, Accounting, or Finance; however, members of the IT area also can sponsor or cosponsor a project.

The size or scope of the project determines the kind of sponsor that is needed. A small, departmental system may only require sponsorship from a single manager; however, a large, organizational initiative may need support from the entire senior management team, and even the CEO. If a project is purely technical in nature (e.g., improvements to the existing IT infrastructure; research into the viability of an emerging

YOUR TURN

3-1 Identify Tangible and Intangible Value

Dominion Virginia Power is one of the nation's ten largest investor-owned electric utilities. The company delivers power to more than two million homes and businesses in Virginia and North Carolina. In 1997, the company overhauled some of its core processes and technology. The goal was to improve customer service and cut operations costs by developing a new workflow and geographic information system. When the project was finished, service engineers who used to sift through thousands of paper maps could pinpoint the locations of electricity poles with computerized searches. The project helped the utility improve management of all its facilities, records, maps, scheduling, and human resources. That, in turn, helped increase employee productivity, improve customer response times, and reduce the costs of operating crews.

Questions:

1. What kinds of things does Dominion Virginia Power do that requires it to know power pole locations? How often does it do these things? Who benefits if the company can locate power poles faster?
2. Based on your answers to question 1, describe three tangible benefits that the company can receive from its new computer system. How can these be quantified?
3. Based on your answers to question 1, describe three intangible benefits that the company can receive from its new computer system. How can these be quantified?

Source: *Computerworld* (November 11, 1997).

technology), then sponsorship from IT is appropriate. When projects have great importance to the business, yet are technically complex, joint sponsorship by both the business and IT may be necessary.

The business need drives the high-level *business requirements* for the system. Requirements are what the information system will do, or what *functionality* it will contain. They need to be explained at a high level so that the approval committee and, ultimately, the project team understand what the business expects from the final product. Business requirements are what features and capabilities the information system will have to include, such as the ability to collect customer orders online or the ability for suppliers to receive inventory information as orders are placed and sales are made.

The project sponsor also should have an idea of the *business value* to be gained from the system, both in tangible and intangible ways. *Tangible value* can be quantified and measured easily (e.g., 2 percent reduction in operating costs). An *intangible* value results from an intuitive belief that the system provides important, but hard-to-measure benefits to the organization (e.g., improved customer service, a better competitive position).

Once the project sponsor identifies a project that meets an important business need, and he or she can identify the system's business requirements and value, it is time to formally initiate the project. In most organizations, project initiation begins with a technique called a system request.

System Request

A *system request* is a document that describes the business reasons for building a system and the value that the system is expected to provide. The project sponsor usually completes this form as part of a formal system project selection process within the organization. Most system requests include five elements: project sponsor, business need, business requirements, business value, and special issues (see Figure 3-1). The sponsor describes the person who will serve as the primary contact for the project, and the business need presents the reasons prompting the project. The business requirements of the project refer to the business capabilities that the system will need to have, and the business value describes the benefits that the organization should expect from the system. *Special issues* are included on the document as a catchall for other information that should be considered in assessing the project. For example, the project may need to be completed by a specific deadline. Project teams need to be aware of any special circumstances that could affect the outcome of the system.

The completed system request is submitted to the *approval committee* for consideration. This approval committee could be a company steering committee that meets regularly to make information systems decisions, a senior executive who has control of organizational resources, or any other decision-making body that governs the use of business investments. The committee reviews the system request and makes an initial determination, based on the information provided, of whether to investigate the proposal or not. If so, the next step is to conduct a feasibility analysis.

Applying the Concepts at CD Selections

Throughout the book, we will apply the concepts in each chapter to a fictitious company called CD Selections. For example, in this section, we will illustrate the creation of a system request. CD Selections is a chain of fifty music stores located in California, with headquarters in Los Angeles. Annual sales last year were \$50 million, and they have been growing at about 3 percent to 5 percent per year for the past few years.

Element	Description	Examples
Project Sponsor	The person who initiates the project and who serves as the primary point of contact for the project on the business side.	Several members of the Finance department Vice President of Marketing IT Manager Steering committee CIO CEO
Business Need	The business-related reason for initiating the system.	Increase sales Improve market share Improve access to information Improve customer service Decrease product defects Streamline supply acquisition Processes
Business Requirements	The business capabilities that the system will provide.	Provide online access to information Capture customer demographic information Include product search capabilities Produce management reports Include online user support
Business Value	The benefits that the system will create for the organization.	3 percent increase in sales 1 percent increase in market share Reduction in headcount by 5 FTEs* \$200,000 cost savings from decreased supply costs \$150,000 savings from removal of existing system
Special Issues or Constraints	Issues that are relevant to the implementation of the system and committee make decisions about the project.	Government-mandated deadline for May 30 System needed in time for the Christmas holiday season Top-level security clearance needed by project team to work with data

* = Full-time equivalent

FIGURE 3-1
Elements of the System Request Form

Background Margaret Mooney, Vice President of Marketing, has recently become both excited by and concerned with the rise of Internet sites selling CDs. The Internet has great potential, but Margaret wants to use it in the right way. Rushing into e-commerce without considering things like its effect on existing brick-and-mortar stores and the implications on existing systems at CD Selections could cause more harm than good.

CD Selections currently has a Web site that provides basic information about the company and about each of its stores (e.g., map, operating hours, phone number). The page was developed by an Internet consulting firm and is hosted by a prominent local Internet Service Provider (ISP) in Los Angeles. The IT department at CD Selections has become experienced with Internet technology as it has worked with the ISP to maintain the site; however, it still has a lot to learn when it comes to conducting business over the Web.

System Request At CD Selections, new IT projects are reviewed and approved by a project steering committee that meets quarterly. The committee has representatives from IT as well as from the major areas of the business. For Margaret, the first step was to prepare a system request for the committee.

CONCEPTS

IN ACTION

3-B Interview with Don Hallacy, President, Technology Services, Sprint Corporation

At Sprint, network projects originate from two vantage points—IT and the business units. IT projects usually address infrastructure and support needs. The business-unit projects typically begin after a business need is identified locally, and a business group informally collaborates with IT regarding how a solution can be delivered to meet customer expectations.

Once an idea is developed, a more formal request process begins, and an analysis team is assigned to investigate and validate the opportunity. This team includes members from the user community and IT, and they scope out at a high level what the project will do; create estimates for technology, training, and development costs; and create a

business case. This business case contains the economic value-add and the net present value of the project.

Of course, not all projects undergo this rigorous process. The larger the project, the more time is allocated to the analysis team. It is important to remain flexible and not let the process consume the organization. At the beginning of each budgetary year, specific capital expenditures are allocated for operational improvements and maintenance. Moreover, this money is set aside to fund quick projects that deliver immediate value without going through the traditional approval process.

—Don Hallacy

Figure 3-2 shows the system request she prepared. The sponsor is Margaret, and the business needs are to increase sales and to better service retail customers. Notice that the need does not focus on the technology, such as the need “to upgrade our Web page.” The focus is on the business aspects: sales and customer service.

For now, the business requirements are described at a very high level of detail. In this case, Margaret’s vision for the requirements includes the ability to help brick-and-mortar stores reach out to new customers. Specifically, customers should be able to search for products over the Internet, locate a retail store that contains the product, put a product on “hold” for later store pickup, and order products that are not currently being stocked.

The business value describes how the requirements will affect the business. Margaret found identifying intangible business value to be fairly straightforward in this case. The Internet is a “hot” area, so she expects the Internet to improve customer recognition and satisfaction. Estimating tangible value is more difficult. She expects that Internet ordering will increase sales in the retail stores, but by how much?

Margaret decides to have her marketing group do some market research to learn how many retail customers do not complete purchases because the store does not carry the item they are looking for. They learn that stores lose approximately 5 percent of total sales from “out-of-stocks and nonstocks.” This number gives Margaret some idea of how much sales could increase from the existing customer base (i.e., about \$50,000 per store), but it does not indicate how many new customers the system will generate.

Estimating how much revenue CD Selections should anticipate from new Internet customers was not simple. One approach was to use some of CD Selections’ standard models for predicting sales of new stores. Retail stores average about \$1 million in sales per year (after they have been open a year or two), depending upon location factors such as city population, average incomes, proximity to universities, and so on. Margaret estimated that adding the new Internet site would have similar effects of adding a new store. This would suggest ongoing revenues of \$1 million, give or take several hundred thousand dollars, after the Web site had been operating for a few years.

Together, the sales from existing customer (\$2.5 million) and new customers (\$1 million) totaled approximately \$3.5 million. Margaret created conservative and optimistic

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 3-2
System Request for CD Selections

System Request—Internet order project	
Project sponsor:	Margaret Mooney, Vice President of Marketing
Business Need:	This project has been initiated to reach new Internet customers and to better serve existing customers using Internet sales support.
Business Requirements:	
Using the Web, customers should be able to search for products and identify the brick-and-mortar stores that have them in stock. They should be able to put items on hold at a store location or place an order for items that are not carried or not in stock. The functionality that the system should have is listed below:	
<ul style="list-style-type: none"> • Search through the CD Selections' inventory of products • Identify the retail stores that have the product in stock • Put a product on hold at a retail store and schedule a time to pick up the product • Place an order for products not currently in stock or not carried by CD Selections • Receive confirmation that an order can be placed and when it will be in stock 	
Business Value:	
We expect that CD Selections will increase sales by reducing lost sales due to out-of-stock or nonstocked items and by reaching out to new customers through its Internet presence. We expect the improved services will reduce customer complaints, primarily because 50 percent of all customer complaints stem from out of stocks or nonstocked items. Also, CD Selections should benefit from improved customer satisfaction and increased brand recognition due to its Internet presence.	
Conservative estimates of tangible value to the company includes:	
<ul style="list-style-type: none"> • \$750,000 in sales from new customers • \$1,875,000 in sales from existing customers • \$50,000 yearly reduction in customer service calls 	
Special Issues or Constraints:	
<ul style="list-style-type: none"> • The Marketing Department views this as a strategic system. This Internet system will add value to our current business model, and it also will serve as a proof of concept for future Internet endeavors. For example, in the future, CD Selections may want to sell products directly over the Internet. • The system should be in place for the holiday shopping season next year. 	

estimates by reducing and increasing this figure by 25 percent. This created a possible range of values from \$2,625,000 to \$4,375,000. Margaret is conservative, so she decided to include the lower number as her sales projection. See Figure 3-2 for the completed system request.

FEASIBILITY ANALYSIS

Once the need for the system and its business requirements have been defined, it is time to create a more detailed business case to better understand the opportunities and limitations associated with the proposed project. *Feasibility analysis* guides the organization in determining whether to proceed with a project. Feasibility analysis also identifies the

**YOUR
TURN**

3-2 Create a System Request

Think about your own university or college and choose an idea that could improve student satisfaction with the course enrollment process. Currently can students enroll for classes from anywhere? How long does it take? Are directions simple to follow? Is online help available?

Next, think about how technology can help support your idea. Would you need completely new technology? Can the current system be changed?

Question:

1. Create a system request that you could give to the administration that explains the sponsor, business need, business requirements, and potential value of the project. Include any constraints or issues that should be considered.

important *risks* associated with the project that must be addressed if the project is approved. As with the system request, each organization has its own process and format for the feasibility analysis, but most include three techniques: technical feasibility, economic feasibility, and organizational feasibility. The results of these techniques are combined into a *Feasibility Study* deliverable that is given to the approval committee at the end of Project Initiation. See Figure 3-3.

Although we will discuss feasibility analysis now within the context of Project Initiation, most project teams will revise their feasibility study throughout the SDLC and revisit its contents at various checkpoints during the project. If at any point the project's risks and limitations outweigh its benefits, the project team may decide to cancel the project or make necessary improvements.

Technical Feasibility: Can We Build It?

- Familiarity with Application: Less familiarity generates more risk
- Familiarity with Technology: Less familiarity generates more risk
- Project Size: Large projects have more risk
- Compatibility: The harder it is to integrate the system with the company's existing technology, the higher the risk

Economic Feasibility: Should We Build It?

- Development costs
- Annual operating costs
- Annual benefits (cost savings and revenues)
- Intangible costs and benefits

Organizational Feasibility: If We Build It, Will They Come?

- Project champion(s)
- Senior management
- Users
- Other stakeholders
- Is the project strategically aligned with the business?

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 3-3
Feasibility Analysis
Assessment Factors

Technical Feasibility

The first technique in the feasibility analysis is to assess the *technical feasibility* of the project, the extent to which the system can be successfully designed, developed, and installed by the IT group. Technical feasibility analysis is in essence a *technical risk analysis* that strives to answer the question: “*Can we build it?*¹

There are many risks that can endanger the successful completion of the project. First and foremost is the users’ and analysts’ *familiarity with the application*. When analysts are unfamiliar with the business application area, they have a greater chance of misunderstanding the users or missing opportunities for improvement. The risks increase dramatically when the users themselves are less familiar with an application, such as with the development of a system to support a new business innovation (e.g., Microsoft starting up a new Internet dating service). In general, the development of new systems is riskier than extensions to an existing system because existing systems tend to be better understood.

Familiarity with the technology is another important source of technical risk. When a system will use technology that has not been used before *within the organization*, there is a greater chance that problems will occur and delays will be incurred because of the need to learn how to use the technology. Risk increases dramatically when the technology itself is new (e.g., a new Java development toolkit).

Project size is an important consideration, whether measured as the number of people on the development team, the length of time it will take to complete the project, or the number of distinct features in the system. Larger projects present more risk, both because they are more complicated to manage and because there is a greater chance that some important system requirements will be overlooked or misunderstood. The extent to which the project is highly integrated with other systems (which is typical of large systems) can cause problems because complexity is increased when many systems must work together.

Finally, project teams need to consider the *compatibility* of the new system with the technology that already exists in the organization. Systems rarely are built in a vacuum—they are built in organizations that have numerous systems already in place. New technology and applications need to be able to integrate with the existing environment for many reasons. They may rely on data from existing systems, they may produce data that feed other applications, and they may have to use the company’s existing communications infrastructure. A new CRM system, for example, has little value if it does not use customer data found across the organization, in existing sales systems, marketing applications, and customer service systems.

The assessment of a project’s technical feasibility is not cut-and-dried because in many cases, some interpretation of the underlying conditions is needed (e.g., how large does a project need to grow before it becomes less feasible?). One approach is to compare the project under consideration with prior projects undertaken by the organization. Another option is to consult with experienced IT professionals in the organization or external IT consultants; often they will be able to judge whether a project is feasible from a technical perspective.

Economic Feasibility

The second element of a feasibility analysis is to perform an *economic feasibility* analysis (also called a *cost–benefit analysis*) that identifies the financial risk associated with the project. This attempts to answer the question: “*Should we build the system?*” Economic feasibility is determined by identifying costs and benefits associated with the system, assigning values to them, and then calculating the cash flow and return on investment for the project. The more expensive the project, the more rigorous and detailed the analysis. Figure 3-4 lists the steps to perform a cost–benefit analysis; each step will be described in the upcoming sections.

¹ We use the words “build it” in the broadest sense. Organizations can also choose to buy a commercial software package and install it, in which case, the question might be: “*Can we select the right package and successfully install it?*”

Identify Costs and Benefits The first task when developing an economic feasibility analysis is to identify the kinds of costs and benefits the system will have and list them along the left-hand column of a spreadsheet. Figure 3-5 lists examples of costs and benefits that may be included.

1. Identify Costs and Benefits	List the tangible costs and benefits for the project. Include both one-time and recurring costs.
2. Assign Values to Costs and Benefits	Work with business users and IT professionals to create numbers for each of the costs and benefits. Even intangibles should be valued if at all possible.
3. Determine Cash Flow	Project what the costs and benefits will be over a period of time, usually three to five years. Apply a growth rate to the numbers, if necessary.
4. Determine Net Present Value	Calculate what the value of future costs and benefits are if measured by today's standards. You will need to select a rate of growth to apply the NPV formula.
5. Determine Return on Investment	Calculate how much money the organization will receive in return for the investment it will make using the ROI formula.
6. Calculate Break-Even Point	Find the first year in which the system has greater benefits than costs. Apply the break-even formula using figures from that year. This will help you understand how long it will take before the system creates real value for the organization.
7. Graph Break-Even Point	Plot the yearly costs and benefits on a line graph. The point at which the lines cross is the break-even point.

FIGURE 3-4
Steps to Conduct Economic Feasibility

Development Costs	Operational Costs
Development Team Salaries	Software Upgrades
Consultant Fees	Software Licensing Fees
Development Training	Hardware Repairs
Hardware and Software	Hardware Upgrades
Vendor Installation	Operational Team Salaries
Office Space and Equipment	Communications Charges
Data Conversion Costs	User Training
Tangible Benefits	Intangible Benefits
Increased Sales	Increased Market Share
Reductions in Staff	Increased Brand Recognition
Reductions in Inventory	Higher Quality Products
Reductions in IT Costs	Improved Customer Service
Better Supplier Prices	Better Supplier Relations

FIGURE 3-5
Example Costs and Benefits for Economic Feasibility

The costs and benefits can be broken down into four categories (1) development costs, (2) operational costs, (3) tangible benefits, and (4) intangibles. *Development costs* are those tangible expenses that are incurred during the construction of the system, such as salaries for the project team, hardware and software expenses, consultant fees, training, and office space and equipment. Development costs are usually thought of as one-time costs. *Operational costs* are those tangible costs that are required to operate the system, such as the salaries for operations staff, software licensing fees, equipment upgrades, and communications charges. Operational costs are usually thought of as ongoing costs.

Revenues and cost savings are those *tangible benefits* that the system enables the organization to collect or *tangible expenses* that the system enables the organization to avoid. Tangible benefits may include increased sales, reductions in staff, and reductions in inventory.

Of course, a project also can affect the organization's bottom line by reaping *intangible benefits* or incurring *intangible costs*. Intangible costs and benefits are more difficult to incorporate into the economic feasibility because they are based on intuition and belief rather than "hard numbers." Nonetheless, they should be listed in the spreadsheet along with the tangible items.

Assign Values to Costs and Benefits Once the types of costs and benefits have been identified, you will need to assign specific dollar values to them. This may seem impossible—how can someone quantify costs and benefits that haven't happened yet? And how can those predictions be realistic? Although this task is very difficult, you have to do the best you can to come up with reasonable numbers for all of the costs and benefits. Only then can the approval committee make an educated decision about whether or not to move ahead with the project.

The best strategy for estimating costs and benefits is to rely on the people who have the best understanding of them. For example, costs and benefits that are related to the technology or the project itself can be provided by the company's IT group or external consultants, and business users can develop the numbers associated with the business (e.g., sales projections, order levels). You also can consider past projects, industry reports, and vendor information, although these approaches probably will be a bit less accurate. Likely, all of the estimates will be revised as the project proceeds.

What about the *intangible* costs and benefits? Sometimes, it is acceptable to list intangible benefits, such as improved customer service, without assigning a dollar value; whereas, other times estimates have to be made regarding how much an intangible benefit is "worth." We suggest

CONCEPTS

3-C Intangible Value at Carlson Hospitality

IN ACTION

I conducted a case study at Carlson Hospitality, a global leader in hospitality services, encompassing more than 1,300 hotel, resort, restaurant, and cruise ship operations in 79 countries. One of its brands, Radisson Hotels & Resorts, researched guest stay information and guest satisfaction surveys. The company was able to quantify how much of a guest's lifetime value can be attributed to his or her perception of the stay experience. As a result, Radisson knows how much of the collective future value of the enterprise is at stake given the perceived quality of stay experience. Using this model, Radisson can confidently show that a 10 percent

increase in customer satisfaction among the 10 percent of highest quality customers, will capture one-point market share for the brand. Each point in market share for the Radisson brand is worth \$20 million in additional revenue.

—Barbara Wixom

Question:

- How can a project team use this information to help determine the economic feasibility of a system?

that you quantify intangible costs or benefits if at all possible. If you do not, how will you know if they have been realized? Suppose that a system is supposed to improve customer service. This is intangible, but let's assume that the greater customer service will decrease the number of customer complaints by 10 percent each year over three years and that \$200,000 is spent on phone charges and phone operators who handle complaint calls. Suddenly we have some very tangible numbers with which to set goals and measure the original intangible benefit.

Figure 3-6 shows costs and benefits along with assigned dollar values. Notice that the customer service intangible benefit has been quantified based on fewer customer complaint phone calls. The intangible benefit of being able to offer services that competitors currently offer was not quantified, but it was listed so that the approval committee will consider the benefit when assessing the system's economic feasibility.

Determine Cash Flow A formal cost–benefit analysis usually contains costs and benefits over a selected number of years (usually three to five years) to show cash flow over time (see Figure 3-7). When using this *cash flow method*, the years are listed across the top of the spreadsheet to represent the time period for analysis, and numeric values are entered in the appropriate cells within the spreadsheet's body. Sometimes fixed amounts are entered into the columns. For example, Figure 3-7 lists the same amount for customer complaint calls and inventory costs for all five years. Usually amounts are augmented by some rate of growth to adjust for inflation or business

Benefits^a	
Increased sales	500,000
Improved customer service ^b	70,000
Reduced inventory costs	68,000
Total benefits	638,000
 Development costs	
2 servers @ \$125,000	250,000
Printer	100,000
Software licenses	34,825
Server software	10,945
Development labor	1,236,525
Total development costs	1,632,295
 Operational costs	
Hardware	54,000
Software	20,000
Operational labor	111,788
Total operational costs	185,788
 Total costs	1,818,083

^a An important yet intangible benefit will be the ability to offer services that our competitors currently offer.

^b Customer service numbers have been based on reduced costs for customer complaint phone calls.

FIGURE 3-6
Assign Values to Costs
and Benefits

	2003	2004	2005	2006	2007	Total
Increased sales	500,000	530,000	561,800	595,508	631,238	
Reduction in customer complaint calls	70,000	70,000	70,000	70,000	70,000	
Reduced inventory costs	68,000	68,000	68,000	68,000	68,000	
Total Benefits:	638,000	668,000	699,800	733,508	769,238	
PV of Benefits:	619,417	629,654	640,416	651,712	663,552	3,204,752
PV of All Benefits:	619,417	1,249,072	1,889,488	2,541,200	3,204,752	
2 Servers @ \$125,000	250,000	0	0	0	0	
Printer	100,000	0	0	0	0	
Software licenses	34,825	0	0	0	0	
Server software	10,945	0	0	0	0	
Development labor	1,236,525	0	0	0	0	
Total Development Costs:	1,632,295	0	0	0	0	
Hardware	54,000	81,261	81,261	81,261	81,261	
Software	20,000	20,000	20,000	20,000	20,000	
Operational labor	111,788	116,260	120,910	125,746	130,776	
Total Operational Costs:	185,788	217,521	222,171	227,007	232,037	
Total Costs:	1,818,083	217,521	222,171	227,007	232,037	
PV of Costs:	1,765,129	205,034	203,318	201,693	200,157	2,575,331
PV of All Costs:	1,765,129	1,970,163	2,173,481	2,375,174	2,575,331	
Total Project Benefits—Costs:	(1,180,083)	450,479	477,629	506,501	537,201	
Yearly NPV:	(1,145,712)	424,620	437,098	450,019	463,395	629,421
Cumulative NPV:	(1,145,712)	(721,091)	(283,993)	166,026	629,421	
Return on Investment:	24.44%	(629,421/2,575,331)				
Break-even Point:	3.63 years	(break-even occurs in year 4; [450,019 – 166,026] / 450,019 = 0.63)				
Intangible Benefits:	This service is currently provided by competitors Improved customer satisfaction					

FIGURE 3-7 Cost-Benefit Analysis

improvements, as shown by the 6 percent increase that is added to the sales numbers in the sample spreadsheet. Finally, totals are added to determine what the overall benefits will be, and the higher the overall total, the more feasible the solution is in terms of its economic feasibility.

Determine Net Present Value and Return on Investment There are several problems with the cash flow method because it does not consider the time value of money (i.e., a dollar today is not worth a dollar tomorrow), and it does not show the overall “bang for the buck” that the organization is receiving from its investment. Therefore, some project teams add additional calculations to the spreadsheet to provide the approval committee with a more accurate picture of the project’s worth.

Net present value (NPV) is used to compare the present value of future cash flows with the investment outlay required to implement the project. Consider the table in Figure 3-8 that shows the future worth of a dollar investment today, given different numbers of years

Number of years	6%	10%	15%
1	0.943	0.909	0.870
2	0.890	0.826	0.756
3	0.840	0.751	0.572
4	0.792	0.683	0.497

This table shows how much a dollar today is worth 1–4 years from now in today's terms across different interest rates.

FIGURE 3-8
The Value of a Future
Dollar Today

and different rates of change. If you have a friend who owes you a dollar today, but instead she gives you a dollar three years from now—you've been had! Given a 10 percent increase in value, you'll be receiving the equivalent of 75 cents in today's terms.

NPV can be calculated in many different ways, some of which are extremely complex. See Figure 3-9 for a basic calculation that can be used in your cash flow analysis to get more relevant values. In Figure 3-7, the present value of the costs and benefits are calculated first (i.e., they are shown at a discounted rate). Then NPV is calculated, and it shows the discounted rate of the combined costs and benefits.

The *return on investment (ROI)* is a calculation that is listed somewhere on the spreadsheet that measures the amount of money an organization receives in return for the money it spends—a high ROI results when benefits far outweigh costs. ROI is determined by taking the total benefits less the costs of the system and dividing that number by the total costs of the system (see Figure 3-9). ROI can be determined per year, or for the entire project over a period of time. One drawback of ROI is that it only considers the end points of the investment, not the cash flow in between, so it should not be used as the sole indicator of a project's worth. Find the ROI figure on the spreadsheet in Figure 3-7.

Calculation	Definition	Formula
Present Value (PV)	The amount of an investment today compared to that same amount in the future, taking into account inflation and time.	$\frac{\text{Amount}}{(1 + \text{interest rate})^n}$ <p>n = number of years in future</p>
Net Present Value (NPV)	The present value of benefit less the present value of costs.	PV Benefits – PV Costs
Return on Investment (ROI)	The amount of revenues or cost savings results from a given investment.	$\frac{\text{Total benefits} - \text{Total costs}}{\text{Total costs}}$
Break-Even Point	The point in time at which the costs of the project equal the value it has delivered.	$\frac{\text{Yearly NPV}^* - \text{Cumulative NPV}}{\text{Yearly NPV}^*}$

*Use the Yearly NPV amount from the first year in which the project has a positive cash flow.
Add the above amount to the year in which the project has a positive cash flow.

FIGURE 3-9 Financial Calculations Used For Cost-Benefit Analysis

CONCEPTS

3-D Return on Investment

IN ACTION

In 1996, IDC conducted a study of sixty-two companies that had implemented data warehousing. They found that the implementations generated an average three-year return on investment of 401 percent. The interesting part is that while forty-five companies reported results between 3 percent and 1,838 percent, the range varied from—1,857 percent to as high as 16,000 percent!

Questions:

1. What kinds of reasons would you give for the varying degrees of return on investment?
2. How would you interpret these results if you were a manager thinking about investing in data warehousing for your company?

Determine Break-Even Point If the project team needs to perform a rigorous cost–benefit analysis, they may need to include information about the length of time before the project will break-even, or when the returns will match the amount invested in the project. The greater the time it takes to break-even, the riskier the project. The *break-even point* is determined by looking at the cash flow over time and identifying the year in which the benefits are larger than the costs (see Figure 3-7). Then, the yearly and cumulative NPV for that year are divided by the yearly NPV to determine how far into the year the break-even will occur. See Figure 3-9 for the break-even calculation.

The break-even point also can be depicted graphically as shown in Figure 3-10. The cumulative present value of the costs and benefits for each year are plotted on a line graph, and the point at which the lines cross is the break-even point.

Alternatives to Traditional Cost–Benefit Analysis There have been concerns raised as to the appropriateness of using traditional cost–benefit analysis using NPV and ROI to determine economic feasibility of an IT project. One of the major problems of using traditional cost–benefit analysis to determine the economic feasibility of an IT investment is that traditional cost–benefit analysis is based on the assumption that the investor must either invest now or not invest at all. However, in most IT investment decisions, the decision to invest is not a now-or-never decision. In most situations, an information system is already in place. As such, the decision to replace or upgrade the current information system can usually be delayed. There have been different proposals

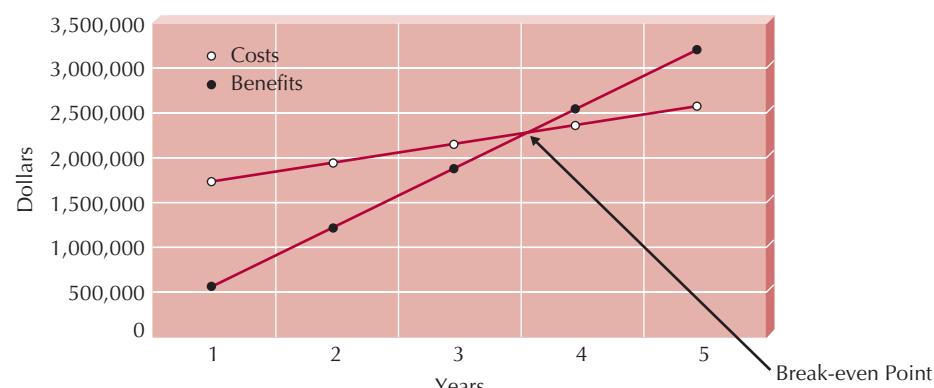


FIGURE 3-10
Break-even Graph

made to overcome some of the weaknesses in traditional cost-benefit analysis.² In this section, we describe the primary alternative that has been proposed for object-oriented systems: *option pricing models (OPMs)*.³

At this point in time, OPMs have had limited use in economic feasibility analysis for IT investment decisions in industry. In fact, there is some controversy as to whether an instrument created for a traded asset (stock) can be used in evaluating IT investment opportunities. However, the preliminary research results demonstrate that their use in IT investment evaluations may be warranted. OPMs have shown promise in evaluating the potential future value of an investment in IT. In many cases in which traditional cost–benefit analysis of investments in IT have predicted that the investment would be a failure, OPMs have shown that they may indeed be feasible.

With object-oriented systems, where classes are designed not only for the current application, but also for use in future development efforts, an investment in developing a class or a set of classes may pay “dividends” well beyond the original system development effort. Furthermore, with the iterative and incremental development emphasis in the object-oriented systems development approaches, such as those described in Chapter 2, an object-oriented project can be viewed as a sequence of smaller projects. As such, you might treat investments in an object-oriented project much as you would an investment in a call option in finance. A *call option* is essentially a contract that gives the right to purchase an amount of stock for a given price for a specified period of time to the purchaser of the call option. However, a call option does not create an obligation to buy the stock.

Treating an IT investment as a call option allows management, at a relevant point in the future, to determine whether additional investment into the evolving system is reasonable. This gives management the flexibility to determine the economic feasibility of a project to decide whether to continue with the development as planned, to abandon the project, to expand the scope of the project, to defer future development, or to shrink the current development effort. In many ways, treating IT investments as call options simply allows management to delay investment decisions until more information is available.

Once the decision is made to invest (i.e., the call option is exercised) the decision is considered irreversible. The idea of irreversible decisions is one of the fundamental assumptions on which OPMs are based. This assumption fits quite well with modern object-oriented system development approaches, in which, once an iteration has begun, an increment is completed before another investment decision is made.

Researchers have studied many different OPMs in terms of their applicability to IT investment.⁴ However, all OPMs share a common thread: both the direct benefit of the proposed

² See, for example, Q. Hu, R. Plant, and D. Hertz, “Software Cost Estimation Using Economic Production Models,” *Journal of MIS* 15(1) (Summer 1998), pp. 143–163; and G. Ooi and C. Soh, “Developing an Activity-based Approach for System Development and Implementation,” *ACM Data Base for Advances in Information Systems* 34(3) (Summer 2003), pp. 54–71.

³ For more information regarding the use of option pricing models in evaluating economic feasibility of information systems, see M. Benaroch and R. Kauffman, “A Case for Using Real Options Pricing Analysis to Evaluate Information Technology Project Investments,” *Information Systems Research* 10(1) (March 1999), pp. 70–86; M. Benaroch and R. Kauffman, “Justifying Electronic Banking Network Expansion Using Real Options Analysis,” *MIS Quarterly* 24(2) (June 2000), pp. 197–225; Q. Dai, R. Kauffman, and S. March, “Analyzing Investments in Object-Oriented Middleware,” *Ninth Workshop on Information Technologies and Systems* (December 1999), pp. 45–50; A. Kambil, J. Henderson, and H. Mohsenzadeh, “Strategic Management of Information Technology Investments: An Options Perspective,” in R.D. Banker, R.J. Kauffman, and M.A. Mahmood (eds.), *Strategic Information Technology Management: Perspectives on Organizational Growth and Competitive Advantage*, Idea Group, 1993; A. Taudes, “Software Growth Options,” *Journal of MIS* 15(1), (Summer 1998), pp. 165–185; A. Taudes, M. Feurstein, and A. Mild, “Options Analysis of Software Platform Decisions: A Case Study,” *MIS Quarterly* 24(2) (June 2000), pp. 227–243.

⁴ Two of the more important OPMs used for evaluating IT investments are the binomial OPM and the Black-Scholes OPM. For more information on these models see J.C. Hull, *Options, Futures, and Other Derivative Securities* (Englewood Cliffs, NJ: Prentice-Hall, 1993).

project and the indirect value (option value) must be computed to determine economic feasibility of an IT investment using an OPM. The direct benefit can be computed using the traditional NPV while the value of the option can be computed using one of the OPMs in the literature. Given that the minimum expected value of an option is always zero, the minimum estimated value for investing using an OPM will be the same as the value given by the traditional approach. However, when the expected value of the option (e.g., future iterations or projects) exceeds zero, an OPM will give an estimated value greater than the traditional approach. The actual calculation of the value of an option is quite complex and, as such, is beyond the scope of this book. However, given how well the OPMs fit the object-oriented systems development approaches, it seems reasonable that OPMs should be considered as alternatives to evaluating IT investments in object-oriented systems.

Organizational Feasibility

The final technique used for feasibility analysis is to assess the *organizational feasibility* of the system, how well the system ultimately will be accepted by its users and incorporated into the ongoing operations of the organization. There are many organizational factors that can have an impact on the project, and seasoned developers know that organizational feasibility can be the most difficult feasibility dimension to assess. In essence, an organizational feasibility analysis attempts to answer the question: “If we build it, will they come?”

One way to assess the organizational feasibility of the project is to understand how well the goals of the project align with business objectives. *Strategic alignment* is the fit between the project and business strategy—the greater the alignment, the less risky the project will be from an organizational feasibility perspective. For example, if the Marketing Department has decided to become more customer focused, then a CRM project that produces integrated customer information would have strong strategic alignment with Marketing’s goal. Many IT projects fail when the IT department initiates them because there is little or no alignment with business unit or organizational strategies.

A second way to assess organizational feasibility is to conduct a *stakeholder analysis*.⁵ A *stakeholder* is a person, group, or organization that can affect (or will be affected by) a new system. In general, the most important stakeholders in the introduction of a new system are the project champion, system users, and organizational management (see Figure 3-11), but systems sometimes affect other stakeholders as well. For example, the IS department can be a stakeholder of a system because IS jobs or roles may be changed significantly after its implementation. One key stakeholder outside of the champion, users, and management in Microsoft’s project that embedded Internet Explorer as a standard part of Windows was the U.S. Department of Justice.

The *champion* is a high-level non-IS executive who is usually but not always the person who created the system request. The champion supports the project by providing time, resources (e.g., money), and political support within the organization by communicating the importance of the system to other organizational decision makers. More than one champion is preferable because if the champion leaves the organization, the support could leave as well.

While champions provide day-to-day support for the system, *organizational management* also needs to support the project. Such management support conveys to the rest of the organization the belief that the system will make a valuable contribution and that necessary resources will be made available. Ideally, management should encourage people in the organization to use the system and to accept the many changes that the system will likely create.

⁵ A good book on stakeholder analysis that presents a series of stakeholder analysis techniques is R. O. Mason and I.I. Mitroff, *Challenging Strategic Planning Assumptions: Theory, Cases, and Techniques*, New York, NY: John Wiley & Sons, 1981.

	Role	Techniques for improvement
Champion	A champion: <ul style="list-style-type: none"> • Initiates the project • Promotes the project • Allocates his or her time to project • Provides resources 	<ul style="list-style-type: none"> • Make a presentation about the objectives of the project and the proposed benefits to those executives who will benefit directly from the system • Create a prototype of the system to demonstrate its potential value
Organizational Management	Organizational managers: <ul style="list-style-type: none"> • Know about the project • Budget enough money for the project • Encourage users to accept and use the system 	<ul style="list-style-type: none"> • Make a presentation to management about the objectives of the project and the proposed benefits • Market the benefits of the system using memos and organizational newsletters • Encourage the champion to talk about the project with his or her peers
System Users	Users: <ul style="list-style-type: none"> • Make decisions that influence the project • Perform hands-on activities for the project • Ultimately determine whether the project is successful by using or not using the system 	<ul style="list-style-type: none"> • Assign users official roles on the project team • Assign users specific tasks to perform with clear deadlines • Ask for feedback from users regularly (e.g., at weekly meetings)

FIGURE 3-11 Some Important Stakeholders for Organizational Feasibility

A third important set of stakeholders is the *system users* who ultimately will use the system once it has been installed in the organization. Too often, the project team meets with users at the beginning of a project and then disappears until after the system is created. In this situation, rarely does the final product meet the expectations and needs of those who are supposed to use it because needs change and users become savvier as the project progresses. User participation should be promoted throughout the development process to make sure that the final system will be accepted and used by getting users actively involved in the development of the system (e.g., performing tasks, providing feedback, and making decisions).

The final feasibility study helps organizations make wiser investments regarding IS because it forces project teams to consider technical, economic, and organizational factors that can affect their projects. It protects IT professionals from criticism by keeping the business units educated about decisions and positioned as the leaders in the decision-making process. Remember—the feasibility study should be revised several times during the project at points where the project team makes critical decisions about the system (e.g., before the design begins). It can be used to support and explain the critical choices that are made throughout the SDLC.

Applying the Concepts at CD Selections

The steering committee met and placed the Internet Order project high on its list of projects. A senior systems analyst, Alec Adams, was assigned to help Margaret conduct a feasibility analysis because of his familiarity with CD Selections' sales and distribution systems. He also was an avid user of the Web and had been offering suggestions for the improvement of CD Selections' Web site.

Alec and Margaret worked closely together over the next few weeks on the feasibility analysis. Figure 3-12 presents the executive summary page of the feasibility analysis; the report itself was about ten pages long, and it provided additional detail and supporting documentation.

As shown in Figure 3-12, the project is somewhat risky from a technical perspective. CD Selections has minimal experience with the proposed application and the technology

Internet Order Feasibility Analysis Executive Summary

Margaret Mooney and Alec Adams created the following feasibility analysis for the CD Selections Internet Order System Project. The System Proposal is attached, along with the detailed feasibility study. The highlights of the feasibility analysis are:

Technical Feasibility

The Internet Order System is feasible technically, although there is some risk.

CD Selections' risk regarding familiarity with Internet order applications is high

- The Marketing Department has little experience with Internet-based marketing and sales.
- The IT Department has strong knowledge of the company's existing order systems; however, it has not worked with Web-enabled order systems.
- Hundreds of retailers that have Internet Order applications exist in the marketplace.

CD Selections' risk regarding familiarity with the technology is medium

- The IT Department has relied on external consultants and an Information Service Provider to develop its existing Web environment.
- The IT Department has gradually learned about Web systems by maintaining the current Web site.
- Development tools and products for commercial Web application development are available in the marketplace, although the IT department has little experience with them.
- Consultants are readily available to provide help in this area.

The project size is considered medium risk

- The project team likely will include less than ten people.
- Business user involvement will be required.
- The project timeframe cannot exceed a year because of the Christmas holiday season implementation deadline, and it should be much shorter.

The compatibility with CD Selections' existing technical infrastructure should be good

- The current Order System is a client-server system built using open standards. An interface with the Web should be possible.
- Retail stores already place and maintain orders electronically.
- An Internet infrastructure already is in place at retail stores and at the corporate headquarters.
- The ISP should be able to scale their services to include a new Order System.

Economic Feasibility

A cost-benefit analysis was performed; see attached spreadsheet for details. A conservative approach shows that the Internet Order System has a good chance of adding to the bottom line of the company significantly.

ROI over 3 years: 229 percent

Total benefit after three years: \$3.5 million (adjusted for present value)

Break-even occurs: after 1.7 years

Intangible Costs and Benefits

- Improved customer satisfaction
- Greater brand recognition

Organizational Feasibility

From an organizational perspective, this project has low risk. The objective of the system, which is to increase sales, is aligned well with the senior management's goal of increasing sales for the company. The move to the Internet also aligns with Marketing's goal to become more savvy in Internet marketing and sales.

The project has a project champion, Margaret Mooney, Vice President of Marketing. Margaret is well positioned to sponsor this project and to educate the rest of the senior management team when necessary. To date, much of senior management is aware of and supports the initiative.

The users of the system, Internet consumers, are expected to appreciate the benefits of CD Selections' Web presence. And, management in the retail stores should be willing to accept the system, given the possibility of increased sales at the store level.

Additional Comments:

- The Marketing Department views this as a strategic system. This Internet system will add value to our current business model, and it also will serve as a proof of concept for future Internet endeavors.
- We should consider hiring a consultant with expertise in similar applications to assist with the project.
- We will need to hire new staff to operate the new system, from both the technical and business operations aspects.

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 3-12 Feasibility Analysis for CD Selections

because the ISP had been managing most of the Web site technology to date. One solution may be to hire a consultant with e-commerce experience to work with the IT department and to offer guidance. Further, the new system would have to exchange order information with the company's brick-and-mortar order system. Currently individual retail stores submit orders electronically, so receiving orders and exchanging information with the Internet systems should be possible.

The economic feasibility analysis includes refined assumptions that Margaret made in the system request. Figure 3-13 shows the summary spreadsheet that led to the conclusions on the feasibility analysis. Development costs are expected to be about \$250,000. This is a very rough estimate, as Alec has had to make some assumptions about the amount of time it will take to design and program the system. These estimates will be revised after a detailed workplan has been developed and as the project proceeds.⁶ Traditionally, operating costs include the costs of the computer operations. In this case, CD Selections has had to include the costs of business staff, because they are creating a new business unit, resulting in a total of about \$450,000 each year. Margaret and Alec have decided to use a conservative estimate for revenues although they note the potential for higher returns. This shows that the project can still add significant business value, even if the underlying assumptions prove to be overly optimistic. The spreadsheet was projected over three years, and the ROI and break-even point were included.

The organizational feasibility is presented in Figure 3-12. There is a strong champion, well placed in the organization to support the project. The project originated in the business or functional side of the company, not the IS department, and Margaret has carefully built up support for the project among the senior management team.

This is an unusual system in that the ultimate end users are the consumers external to CD Selections. Margaret and Alec have not done any specific market research to see how well potential customers will react to the CD Selections system, so this is a risk.

An additional stakeholder in the project is the management team responsible for the operations of the traditional stores, and the store managers. They should be quite supportive given the added service that they now can offer. Margaret and Alec need to make sure that they are included in the development of the system so that they can appropriately incorporate it into their business processes.

YOUR TURN

3-3 Create a Feasibility Analysis

Think about the idea that you developed in "Your Turn 3-2" to improve your university or college course enrollment.

Questions:

1. List three things that influence the technical feasibility of the system.

2. List three things that influence the economic feasibility of the system.
3. List three things that influence the organizational feasibility of the system.
4. How can you learn more about the issues that affect the three kinds of feasibility?

⁶ Some of the salary information may seem high to you. Most companies use a "full cost" model for estimating salary cost in which all benefits (e.g., health insurance, retirement, payroll taxes) are included in salaries when estimating costs.

	2003	2004	2005	Total
Increased sales from new customers	0	750,000	772,500	
Increased sales from existing customers	0	1,875,000	1,931,250	
Reduction in customer complaint calls	0	50,000	50,000	
Total Benefits:	<u>0</u>	<u>2,675,000</u>	<u>2,753,750</u>	
PV of Benefits:	<u>0</u>	<u>2,521,444</u>	<u>2,520,071</u>	<u>5,041,515</u>
PV of All Benefits:	<u>0</u>	<u>2,521,444</u>	<u>5,041,515</u>	
Labor: Analysis and Design	42,000	0	0	
Labor: Implementation	120,000	0	0	
Consultant Fees	50,000	0	0	
Training	5,000	0	0	
Office Space and Equipment	2,000	0	0	
Software	10,000	0	0	
Hardware	25,000	0	0	
Total Development Costs:	254,000	0	0	
Labor: Webmaster	85,000	87,550	90,177	
Labor: Network Technician	60,000	61,800	63,654	
Labor: Computer Operations	50,000	51,500	53,045	
Labor: Business Manager	60,000	61,800	63,654	
Labor: Assistant Manager	45,000	46,350	47,741	
Labor: 3 Staff	90,000	92,700	95,481	
Software upgrades	1,000	1,000	1,000	
Software licenses	3,000	1,000	1,000	
Hardware upgrades	5,000	3,000	3,000	
User training	2,000	1,000	1,000	
Communications charges	20,000	20,000	20,000	
Marketing expenses	25,000	25,000	25,000	
Total Operational Costs:	446,000	452,700	464,751	
Total Costs:	<u>700,000</u>	<u>452,700</u>	<u>464,751</u>	
PV of Costs:	<u>679,612</u>	<u>426,713</u>	<u>425,313</u>	<u>1,531,638</u>
PV of all Costs:	<u>679,612</u>	<u>1,106,325</u>	<u>1,531,638</u>	
Total Project Costs Less Benefits:	(700,000)	2,222,300	2,288,999	
Yearly NPV:	(679,612)	2,094,731	2,094,758	3,509,878
Cumulative NPV:	(679,612)	1,415,119	3,509,878	
Return on Investment:	229.16%	(3,509,878/1,531,638)		
Break-even Point:	1.32 years	(break-even occurs in year 2; [2,094,731 – 1,415,119] / 2,094,731 = 0.32)		
Intangible Benefits:	Greater brand recognition Improved customer satisfaction			

FIGURE 3-13 Economic Feasibility Analysis for CD Selections

PROJECT SELECTION

Once the feasibility analysis has been completed, it is submitted back to the approval committee along with a revised system request. The committee then decides whether to approve the project, decline the project, or table it until additional information is available. At the project level, the committee considers the value of the project by examining the business need (found in the system request) and the risks of building the system (presented in the feasibility analysis).

Before approving the project, however, the committee also considers the project from an organizational perspective; it has to keep in mind the company's entire portfolio of projects. This way of managing projects is called *portfolio management*. Portfolio management takes into consideration the different kinds of projects that exist in an organization—large and small, high risk and low risk, strategic and tactical (see Figure 3-14 for the different ways of classifying projects). A good project portfolio will have the most appropriate mix of projects for the organization's needs. The committee acts as portfolio manager with the goal of maximizing the cost/benefit performance and other important factors of the projects in their portfolio. For example, a organization may want to keep high-risk projects to less than 20 percent of its total project portfolio.

The approval committee must be selective about where to allocate resources because the organization has limited funds. This involves *trade-offs* in which the organization must give up something in return for something else to keep its portfolio well balanced. If there are three potentially high-payoff projects, yet all have very high risk, then maybe only one of the projects will be selected. Also, there are times when a system at the project level makes good business sense, but it does not at the organization level. Thus, a project may show a very strong ROI and support important business needs for a part of the company; however, it is not selected. This could happen for many reasons—because there is no money in the budget for another system, the organization is about to go through some kind of change (e.g., a merger, an implementation of a company-wide system like an ERP system), projects that meet the same business requirements already are underway, or the system does not align well with current or future corporate strategy.

Applying the Concepts at CD Selections

The approval committee met and reviewed the Internet Order System project along with two other projects—one that called for the implementation of corporate Intranet and

Size	What is the size? How many people are needed to work on the project?
Cost	How much will the project cost the organization?
Purpose	What is the purpose of the project? Is it meant to improve the technical infrastructure? Support a current business strategy? Improve operations? Demonstrate a new innovation?
Length	How long will the project take before completion? How much time will go by before value is delivered to the business?
Risk	How likely is it that the project will succeed or fail?
Scope	How much of the organization is affected by the system? A department? A division? The entire corporation?
Return on investment	How much money does the organization expect to receive in return for the amount the project costs?

FIGURE 3-14
Ways to Classify Projects

another that proposed in-store kiosks that would provide customers with information about the CDs that the store carried. Unfortunately, the budget would only allow for one project to be approved, so the committee carefully examined the costs, expected benefits, risks, and strategic alignment of all three projects. Currently, a primary focus of upper management is increasing sales in the retail stores, and the Internet system and kiosk project best aligned with that goal. Given that both projects had equal risk, but that the Internet Order project expected a much greater return, the committee decided to fund the Internet Order System.

SUMMARY

Project Initiation

Project initiation is the point at which an organization creates and assesses the original goals and expectations for a new system. The first step in the process is to identify the business value for the system by developing a system request that provides basic information

YOUR

TURN

3-4 To Select or Not To Select

It seems hard to believe that an approval committee would not select a project that meets real business needs, has a high potential ROI, and has a positive feasibility analysis.

Think of a company you have worked for or know about. Describe a scenario in which a project may be very attractive at the project level, but not at the organization level.

CONCEPTS

IN ACTION

3-E Interview with Carl Wilson, CIO, Marriott Corporation

At Marriott, we don't have IT projects—we have business initiatives and strategies that are enabled by IT. As a result, the only time a traditional "IT project" occurs is when we have an infrastructure upgrade that will lower costs or leverage better functioning technology. In this case, IT has to make a business case for the upgrade and prove its value to the company.

The way IT is involved in business projects in the organization is twofold. First, senior IT positions are filled by people with good business understanding. Second, these people are placed on key business committees and forums where the real business happens, such as finding ways to satisfy guests. Because IT has a seat at the table, we are able to spot opportunities to support business strategy. We look for ways in which IT can enable or better support business initiatives as they arise.

Therefore, business projects are proposed, and IT is one component of them. These projects are then evaluated

the same as any other business proposal, such as a new resort—by examining the return on investment and other financial measures.

At the organizational level, I think of projects as must-do's, should-do's, and nice-to-do's. The "must do's" are required to achieve core business strategy, such as guest preference. The "should do's" help grow the business and enhance the functionality of the enterprise. These can be somewhat untested, but good drivers of growth. The "nice-to-do's" are more experimental and look farther out into the future.

The organization's project portfolio should have a mix of all three kinds of projects, with a much greater proportion devoted to the "must-do's."

—Carl Wilson

CONCEPTS**IN ACTION****3-F A Project That Does Not Get Selected**

Hygeia Travel Health is a Toronto-based health insurance company whose clients are the insurers of foreign tourists to the United States and Canada. Its project selection process is relatively straightforward. The project evaluation committee, consisting of six senior executives, splits into two groups. One group includes the CIO, along with the heads of operations and research and development, and it analyzes the costs of every project. The other group consists of the two chief marketing officers and the head of business development, and they analyze the expected benefits. The groups are permanent, and to stay objective, they don't discuss a project until both sides have evaluated it. The results are then shared, both on a spreadsheet and in conversation. Projects are then approved, passed over, or tabled for future consideration.

Last year, the marketing department proposed purchasing a claims database filled with detailed information on the costs of treating different conditions at different facilities. Hygeia was to use this information to estimate how much money insurance providers were likely to owe on a given claim if a patient was treated at a certain hospital as opposed to any other. For example, a 45-year-old man suffering a heart attack may accrue \$5,000 in treatment costs at hospital A, but only \$4,000 at hospital B. This information

would allow Hygeia to recommend the cheaper hospital to its customer. That would save the customer money and help differentiate Hygeia from its competitors.

The benefits team used the same three-meeting process to discuss all the possible benefits of implementing the claims database. Members of the team talked to customers and made a projection using Hygeia's past experience and expectations about future business trends. The verdict: The benefits team projected a revenue increase of \$210,000. Client retention would rise by 2 percent. And overall, profits would increase by 0.25 percent.

The costs team, meanwhile, came up with large estimates: \$250,000 annually to purchase the database and an additional \$71,000 worth of internal time to make the information usable. Put it all together and it was a financial loss of \$111,000 in the first year.

The project still could have been good for marketing—maybe even good enough to make the loss acceptable. But some of Hygeia's clients were also in the claims information business and therefore potential competitors. This, combined with the financial loss, was enough to make the company reject the project.

Source: "Two Teams Are Better Than One" CIO Magazine, July 15, 2001 by Ben Worthen.

**YOUR
TURN****3-5 Project Selection**

In April 1999, one of Capital Blue Cross's healthcare insurance plans had been in the field for three years but hadn't performed as well as expected. The ratio of premiums to claims payments wasn't meeting historic norms. In order to revamp the product features or pricing to boost performance, the company needed to understand why it was underperforming. The stakeholders came to the discussion already knowing they needed better extraction and analysis of usage data in order to understand product shortcomings and recommend improvements.

After listening to input from the user teams, the stakeholders proposed three options. One was to persevere with the current manual method of pulling data from flat files via ad hoc reports and retyping it into spreadsheets.

The second option was to write a program to dynamically mine the needed data from Capital's customer information control system (CICS). While the system was processing claims, for instance, the program would pull

out up-to-the-minute data at a given point in time for users to analyze.

The third alternative was to develop a decision-support system to allow users to make relational queries from a data mart containing a replication of the relevant claims and customer data.

Each of these alternatives was evaluated on cost, benefits, risks, and intangibles.

Question:

1. What are three costs, benefits, risks, and intangibles associated with each project?
2. Based on your answer to question 1, which project would you choose?

Source: "Capital Blue Cross," CIO Magazine, February 15, 2000, by Richard Pastore.

about the proposed system. Next the analysts perform a feasibility analysis to determine the technical, economic, and organizational feasibility of the system, and if appropriate, the system is approved, and the development project begins.

System Request

The business value for an information system is identified and then described using a system request. This form contains the project's sponsor, business need, business requirements, and business value of the information system, along with any other issues or constraints that are important to the project. The document is submitted to an approval committee who determines whether the project would be a wise investment of the organization's time and resources.

Feasibility Analysis

A feasibility analysis is then used to provide more detail about the risks associated with the proposed system, and it includes technical, economic, and organizational feasibilities. The technical feasibility focuses on whether the system *can* be built by examining the risks associated with the users' and analysts' familiarity with the application, familiarity with the technology, and project size. The economic feasibility addresses whether the system *should* be built. It includes a cost–benefit analysis of development costs, operational costs, tangible benefits, and intangible costs and benefits. Finally, the organizational feasibility assesses how well the system will be accepted by its users and incorporated into the ongoing operations of the organization. The strategic alignment of the project and a stakeholder analysis can be used to assess this feasibility dimension.

Project Selection

Once the feasibility analysis has been completed, it is submitted back to the approval committee along with a revised system request. The committee then decides whether to approve the project, decline the project, or table it until additional information is available. The project selection process takes into account all of the projects in the organization using portfolio management. The approval committee weighs many factors and makes trade-offs before a project is selected.

KEY TERMS

Approval committee	Feasibility study	Return on/investment (ROI)
Break-even point	First mover	Risk
Business need	Functionality	Special issues
Business requirement	Intangible benefits	Stakeholder
Business value	Intangible costs	Stakeholder analysis
Call option	Intangible value	Strategic alignment
Cash flow method	Net present value (NPV)	System request
Champion	Operational cost	System users
Compatibility	Option pricing models (OPMs)	Tangible benefits
Cost–benefit analysis	Organizational feasibility	Tangible expenses
Development cost	Organizational management	Tangible value
Economic feasibility	Portfolio management	Technical feasibility
Emerging technology	Project	Technical risk analysis
Familiarity with the application	Project initiation	Trade-offs
Familiarity with the technology	Project size	
Feasibility analysis	Project sponsor	

QUESTIONS

1. Give three examples of business needs for a system.
2. What is the purpose of an approval committee? Who is usually on this committee?
3. Why should the system request be created by a businessperson as opposed to an IS professional?
4. What is the difference between intangible value and tangible value? Give three examples of each.
5. What are the purposes of the system request and the feasibility analysis? How are they used in the project selection process?
6. Describe two special issues that may be important to list on a system request.
7. Describe the three techniques for feasibility analysis.
8. What factors are used to determine project size?
9. Describe a “risky” project in terms of technical feasibility.
10. Describe a project that would not be considered “risky.”
11. List two intangible benefits. Describe how these benefits can be quantified.
12. List two tangible benefits and two operational costs for a system. How would you determine the values that should be assigned to each item?
13. Explain the net present value and return on investment for a cost–benefit analysis. Why would these calculations be used?
14. What is the break-even point for the project? How is it calculated?
15. What is stakeholder analysis? Discuss three stakeholders that would be relevant for most projects.

EXERCISES

- A. Locate a news article in an IT trade magazine (e.g., *Computerworld*) about an organization that is implementing a new computer system. Describe the tangible and intangible value that the organization likely will realize from the new system.
- B. Car dealers have realized how profitable it can be to sell automobiles using the Web. Pretend you work for a local car dealership that is part of a large chain such as CarMax. Create a system request you might use to develop a Web-based sales system. Remember to list special issues that are relevant to the project.
- C. Suppose that you are interested in buying yourself a new computer. Create a cost–benefit analysis that illustrates the return on investment that you would receive from making this purchase. Computer-related Web sites (e.g., Dell Computers, Compaq Computers) should have real tangible costs that you can include in your analysis. Project your numbers out to include a three-year period of time and provide the net present value of the final total.
- D. Consider the Amazon.com Web site. The management of the company decided to extend their Web-based system to include products other than books (e.g., wine, specialty gifts). How would you have assessed the feasibility of this venture when the idea first came up? How “risky” would you have considered the project that implemented this idea? Why?
- E. Interview someone who works in a large organization and ask him or her to describe the approval process that exists for approving new development projects. What do they think about the process? What are the problems? What are the benefits?
- F. Reread the “Your Turn 3-1” box (Identify Tangible and Intangible Value). Create a list of the stakeholders that should be considered in a stakeholder analysis of this project.

MINICASES

1. The Amberssen Specialty Company is a chain of twelve retail stores that sell a variety of imported gift items, gourmet chocolates, cheeses, and wines in the Toronto area. Amberssen has an IS staff of three people who have created a simple but effective information system

of networked point-of-sale registers at the stores, and a centralized accounting system at the company headquarters. Harry Hilman, the head of Amberssen’s IS group, has just received the following memo from Bill Amberssen, Sales Director (and son of Amberssen’s founder).

Harry—it's time Amberssen Specialty launched itself on the Internet. Many of our competitors are already there, selling to customers without the expense of a retail storefront, and we should be there too. I project that we could double or triple our annual revenues by selling our products on the Internet. I'd like to have this ready by Thanksgiving, in time for the prime holiday gift-shopping season. Bill

After pondering this memo for several days, Harry scheduled a meeting with Bill so that he could clarify Bill's vision of this venture. Using the standard content of a system request as your guide, prepare a list of questions that Harry needs to have answered about this project.

2. The Decker Company maintains a fleet of ten service trucks and crews that provide a variety of plumbing, heating, and cooling repair services to residential customers. Currently, it takes on average about six hours before a service team responds to a service request. Each truck and crew averages twelve service calls per week, and the average revenue earned per service call is \$150. Each truck is in service fifty weeks per year. Due to the difficulty in scheduling and routing, there is considerable slack time for each truck and crew during a typical week.

In an effort to more efficiently schedule the trucks and crews and improve their productivity, Decker management is evaluating the purchase of a prewritten routing and scheduling software package. The benefits of the system will include reduced response time to service requests and more productive service teams, but management is having trouble quantifying these benefits.

One approach is to make an estimate of how much service response time will decrease with the new system, which then can be used to project the increase in the number of service calls made each week. For example, if the system permits the average service response time to fall to four hours, management believes that each truck will be able to make sixteen service calls per week on average—an increase of four calls per week. With each truck making four additional calls per week and the average revenue per call at \$150, the revenue increase per truck per week is \$600 ($4 \times \150). With ten trucks in service fifty weeks per year, the average annual revenue increase will be \$300,000 ($\$600 \times 10 \times 50$).

Decker Company management is unsure whether the new system will enable response time to fall to four hours on average, or will be some other number. Therefore, management has developed the following range of outcomes that may be possible outcomes of the new system, along with probability estimates of each outcome occurring.

New Response Time	# Calls/Truck/Week	Likelihood
2 hours	20	20%
3 hours	18	30%
4 hours	16	50%

Given these figures, prepare a spreadsheet model that computes the expected value of the annual revenues to be produced by this new system.

PART ONE

PLANNING PHASE

The Planning Phase is the fundamental process of understanding why an information system should be built, and determining how the project team will build it. The deliverables are combined into a system request which is presented to the project sponsor and approval committee at the end of this phase. They decide whether it is advisable to proceed with the system.

CHAPTER 3

PROJECT INITIATION

This chapter describes Project Initiation, the point at which an organization creates and assesses the original goals and expectations for a new system. The first step in the process is to identify a project that will deliver value to the business and to create a system request that provides basic information about the proposed system. Next the analysts perform a feasibility analysis to determine the technical, economic, and organizational feasibility of the system, and if appropriate, the system is selected, and the development project begins.

OBJECTIVES

- Understand the importance of linking the information system to business needs.
- Be able to create a system request.
- Understand how to assess technical, economic, and organizational feasibility.
- Be able to perform a feasibility analysis.
- Understand how projects are selected in some organizations.

CHAPTER OUTLINE

Introduction	Economic Feasibility
Project Identification	Organizational Feasibility
System Request	Applying the Concepts at CD Selections
Applying the Concepts at CD Selections	Project Selection
Feasibility Analysis	Applying the Concepts at CD Selections
Technical Feasibility	Summary

INTRODUCTION

The first step in any new development project is for someone—a manager, staff member, sales representative, or systems analyst—to see an opportunity to improve the business. New systems start first and foremost from some business need or opportunity. Many ideas for new systems or improvements to existing ones arise from the application of a new technology, but an understanding of technology is usually secondary to a solid understanding of the business and its objectives.

This may sound like common sense, but unfortunately, many projects are started without a clear understanding of how the system will improve the business. The IS field is filled with thousands of buzzwords, fads, and trends (e.g., customer relationship management [CRM], mobile computing, data mining). The promise of these innovations can appear so attractive that organizations begin projects even if they are not sure what value they offer,

because they believe that the technologies are somehow important in their own right. A 1996 survey by the Standish Group found that 42 percent of all corporate IS projects were abandoned before completion; a similar 1996 study by the General Accounting Office found 53 percent of all U.S. government IS projects were abandoned. Most times, problems can be traced back to the very beginning of the SDLC where too little attention was given to the identifying business value and understanding the risks associated with the project.

Does this mean that technical people should not recommend new systems projects? Absolutely not. In fact, the ideal situation is for both IT people (i.e., the experts in systems) and the business people (i.e., the experts in business) to work closely to find ways for technology to support business needs. In this way, organizations can leverage the exciting technologies that are available while ensuring that projects are based upon real business objectives, such as increasing sales, improving customer service, and decreasing operating expenses. Ultimately, information systems need to affect the organization's bottom line (in a positive way!).

In general, a *project* is a set of activities with a starting point and an ending point meant to create a system that brings value to the business. *Project initiation* begins when someone (or some group) in the organization (called the *project sponsor*) identifies some business value that can be gained from using information technology. The proposed project is described briefly using a technique called the system request, which is submitted to an approval committee for consideration. The approval committee reviews the system request and makes an initial determination, based on the information provided, of whether to investigate the proposal or not. If so, the next step is the feasibility analysis.

The feasibility analysis plays an important role in deciding whether to proceed with an IS development project. It examines the technical, economic, and organizational pros and cons of developing the system, and it gives the organization a slightly more detailed picture of the advantages of investing in the system as well as any obstacles that could arise. In most cases, the project sponsor works together with an analyst (or analyst team) to develop the feasibility analysis for the approval committee.

Once the feasibility analysis has been completed, it is submitted back to the approval committee along with a revised system request. The committee then decides whether to approve the project, decline the project, or table it until additional information is available. Projects are selected by weighing risks and return, and by making trade-offs at the organizational level.

CONCEPTS

IN ACTION

3-A Interview with Lyn McDermid, CIO, Dominion Virginia Power

A CIO needs to have a global view when identifying and selecting projects for her organization. I would get lost in the trees if I were to manage on a project-by-project basis. Given this, I categorize my projects according to my three roles as a CIO, and the mix of my project portfolio changes depending on the current business environment.

My primary role is to **keep the business running**. That means every day when each person comes to work, they can perform his or her job efficiently. I measure this using various service level, cost, and productivity measures. Projects that keep the business running could have a high priority if the business were in the middle of a merger, or a low priority if things were running smoothly, and it were "business as usual."

My second role is to push **innovation that creates value for the business**. I manage this by looking at our lines of business and asking which lines of business create the most value for the company. These are the areas for which I should be providing the most value. For example, if we had a highly innovative marketing strategy, I would push for innovation there. If operations were running smoothly, I would push less for innovation in that area.

My third role is strategic, to look beyond today and find **new opportunities** for both IT and the business of providing energy. This may include investigating process systems, such as automated meter reading or looking into the possibilities of wireless technologies.

—Lyn McDermid

PROJECT IDENTIFICATION

A project is identified when someone in the organization identifies a *business need* to build a system. This could occur within a business unit or IT, by a steering committee charged with identifying business opportunities, or evolve from a recommendation made by external consultants. Examples of business needs include supporting a new marketing campaign, reaching out to a new type of customer, or improving interactions with suppliers. Sometimes, needs arise from some kind of “pain” within the organization, such as a drop in market share, poor customer service levels, or increased competition. Other times, new business initiatives and strategies are created, and a system is required to enable them.

Business needs also can surface when the organization identifies unique and competitive ways of using IT. Many organizations keep an eye on *emerging technology*, which is technology that is still being developed and not yet viable for widespread business use. For example, if companies stay abreast of technology like the Internet, smart cards, and scent technology in their earliest stages, they can develop business strategies that leverage the capabilities of these technologies and introduce them into the marketplace as a *first mover*. Ideally, they can take advantage of this first-mover advantage by making money and continuing to innovate while competitors trail behind.

The *project sponsor* is someone who recognizes the strong business need for a system and has an interest in seeing the system succeed. He or she will work throughout the SDLC to make sure that the project is moving in the right direction from the perspective of the business. The project sponsor serves as the primary point of contact for the system. Usually the sponsor of the project is from a business function, such as Marketing, Accounting, or Finance; however, members of the IT area also can sponsor or cosponsor a project.

The size or scope of the project determines the kind of sponsor that is needed. A small, departmental system may only require sponsorship from a single manager; however, a large, organizational initiative may need support from the entire senior management team, and even the CEO. If a project is purely technical in nature (e.g., improvements to the existing IT infrastructure; research into the viability of an emerging

YOUR TURN

3-1 Identify Tangible and Intangible Value

Dominion Virginia Power is one of the nation's ten largest investor-owned electric utilities. The company delivers power to more than two million homes and businesses in Virginia and North Carolina. In 1997, the company overhauled some of its core processes and technology. The goal was to improve customer service and cut operations costs by developing a new workflow and geographic information system. When the project was finished, service engineers who used to sift through thousands of paper maps could pinpoint the locations of electricity poles with computerized searches. The project helped the utility improve management of all its facilities, records, maps, scheduling, and human resources. That, in turn, helped increase employee productivity, improve customer response times, and reduce the costs of operating crews.

Questions:

1. What kinds of things does Dominion Virginia Power do that requires it to know power pole locations? How often does it do these things? Who benefits if the company can locate power poles faster?
2. Based on your answers to question 1, describe three tangible benefits that the company can receive from its new computer system. How can these be quantified?
3. Based on your answers to question 1, describe three intangible benefits that the company can receive from its new computer system. How can these be quantified?

Source: *Computerworld* (November 11, 1997).

technology), then sponsorship from IT is appropriate. When projects have great importance to the business, yet are technically complex, joint sponsorship by both the business and IT may be necessary.

The business need drives the high-level *business requirements* for the system. Requirements are what the information system will do, or what *functionality* it will contain. They need to be explained at a high level so that the approval committee and, ultimately, the project team understand what the business expects from the final product. Business requirements are what features and capabilities the information system will have to include, such as the ability to collect customer orders online or the ability for suppliers to receive inventory information as orders are placed and sales are made.

The project sponsor also should have an idea of the *business value* to be gained from the system, both in tangible and intangible ways. *Tangible value* can be quantified and measured easily (e.g., 2 percent reduction in operating costs). An *intangible* value results from an intuitive belief that the system provides important, but hard-to-measure benefits to the organization (e.g., improved customer service, a better competitive position).

Once the project sponsor identifies a project that meets an important business need, and he or she can identify the system's business requirements and value, it is time to formally initiate the project. In most organizations, project initiation begins with a technique called a system request.

System Request

A *system request* is a document that describes the business reasons for building a system and the value that the system is expected to provide. The project sponsor usually completes this form as part of a formal system project selection process within the organization. Most system requests include five elements: project sponsor, business need, business requirements, business value, and special issues (see Figure 3-1). The sponsor describes the person who will serve as the primary contact for the project, and the business need presents the reasons prompting the project. The business requirements of the project refer to the business capabilities that the system will need to have, and the business value describes the benefits that the organization should expect from the system. *Special issues* are included on the document as a catchall for other information that should be considered in assessing the project. For example, the project may need to be completed by a specific deadline. Project teams need to be aware of any special circumstances that could affect the outcome of the system.

The completed system request is submitted to the *approval committee* for consideration. This approval committee could be a company steering committee that meets regularly to make information systems decisions, a senior executive who has control of organizational resources, or any other decision-making body that governs the use of business investments. The committee reviews the system request and makes an initial determination, based on the information provided, of whether to investigate the proposal or not. If so, the next step is to conduct a feasibility analysis.

Applying the Concepts at CD Selections

Throughout the book, we will apply the concepts in each chapter to a fictitious company called CD Selections. For example, in this section, we will illustrate the creation of a system request. CD Selections is a chain of fifty music stores located in California, with headquarters in Los Angeles. Annual sales last year were \$50 million, and they have been growing at about 3 percent to 5 percent per year for the past few years.

Element	Description	Examples
Project Sponsor	The person who initiates the project and who serves as the primary point of contact for the project on the business side.	Several members of the Finance department Vice President of Marketing IT Manager Steering committee CIO CEO
Business Need	The business-related reason for initiating the system.	Increase sales Improve market share Improve access to information Improve customer service Decrease product defects Streamline supply acquisition Processes
Business Requirements	The business capabilities that the system will provide.	Provide online access to information Capture customer demographic information Include product search capabilities Produce management reports Include online user support
Business Value	The benefits that the system will create for the organization.	3 percent increase in sales 1 percent increase in market share Reduction in headcount by 5 FTEs* \$200,000 cost savings from decreased supply costs \$150,000 savings from removal of existing system
Special Issues or Constraints	Issues that are relevant to the implementation of the system and committee make decisions about the project.	Government-mandated deadline for May 30 System needed in time for the Christmas holiday season Top-level security clearance needed by project team to work with data

* = Full-time equivalent

FIGURE 3-1
Elements of the System Request Form

Background Margaret Mooney, Vice President of Marketing, has recently become both excited by and concerned with the rise of Internet sites selling CDs. The Internet has great potential, but Margaret wants to use it in the right way. Rushing into e-commerce without considering things like its effect on existing brick-and-mortar stores and the implications on existing systems at CD Selections could cause more harm than good.

CD Selections currently has a Web site that provides basic information about the company and about each of its stores (e.g., map, operating hours, phone number). The page was developed by an Internet consulting firm and is hosted by a prominent local Internet Service Provider (ISP) in Los Angeles. The IT department at CD Selections has become experienced with Internet technology as it has worked with the ISP to maintain the site; however, it still has a lot to learn when it comes to conducting business over the Web.

System Request At CD Selections, new IT projects are reviewed and approved by a project steering committee that meets quarterly. The committee has representatives from IT as well as from the major areas of the business. For Margaret, the first step was to prepare a system request for the committee.

CONCEPTS

IN ACTION

3-B Interview with Don Hallacy, President, Technology Services, Sprint Corporation

At Sprint, network projects originate from two vantage points—IT and the business units. IT projects usually address infrastructure and support needs. The business-unit projects typically begin after a business need is identified locally, and a business group informally collaborates with IT regarding how a solution can be delivered to meet customer expectations.

Once an idea is developed, a more formal request process begins, and an analysis team is assigned to investigate and validate the opportunity. This team includes members from the user community and IT, and they scope out at a high level what the project will do; create estimates for technology, training, and development costs; and create a

business case. This business case contains the economic value-add and the net present value of the project.

Of course, not all projects undergo this rigorous process. The larger the project, the more time is allocated to the analysis team. It is important to remain flexible and not let the process consume the organization. At the beginning of each budgetary year, specific capital expenditures are allocated for operational improvements and maintenance. Moreover, this money is set aside to fund quick projects that deliver immediate value without going through the traditional approval process.

—Don Hallacy

Figure 3-2 shows the system request she prepared. The sponsor is Margaret, and the business needs are to increase sales and to better service retail customers. Notice that the need does not focus on the technology, such as the need “to upgrade our Web page.” The focus is on the business aspects: sales and customer service.

For now, the business requirements are described at a very high level of detail. In this case, Margaret’s vision for the requirements includes the ability to help brick-and-mortar stores reach out to new customers. Specifically, customers should be able to search for products over the Internet, locate a retail store that contains the product, put a product on “hold” for later store pickup, and order products that are not currently being stocked.

The business value describes how the requirements will affect the business. Margaret found identifying intangible business value to be fairly straightforward in this case. The Internet is a “hot” area, so she expects the Internet to improve customer recognition and satisfaction. Estimating tangible value is more difficult. She expects that Internet ordering will increase sales in the retail stores, but by how much?

Margaret decides to have her marketing group do some market research to learn how many retail customers do not complete purchases because the store does not carry the item they are looking for. They learn that stores lose approximately 5 percent of total sales from “out-of-stocks and nonstocks.” This number gives Margaret some idea of how much sales could increase from the existing customer base (i.e., about \$50,000 per store), but it does not indicate how many new customers the system will generate.

Estimating how much revenue CD Selections should anticipate from new Internet customers was not simple. One approach was to use some of CD Selections’ standard models for predicting sales of new stores. Retail stores average about \$1 million in sales per year (after they have been open a year or two), depending upon location factors such as city population, average incomes, proximity to universities, and so on. Margaret estimated that adding the new Internet site would have similar effects of adding a new store. This would suggest ongoing revenues of \$1 million, give or take several hundred thousand dollars, after the Web site had been operating for a few years.

Together, the sales from existing customer (\$2.5 million) and new customers (\$1 million) totaled approximately \$3.5 million. Margaret created conservative and optimistic

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 3-2
System Request for CD Selections

System Request—Internet order project	
Project sponsor:	Margaret Mooney, Vice President of Marketing
Business Need:	This project has been initiated to reach new Internet customers and to better serve existing customers using Internet sales support.
Business Requirements:	
Using the Web, customers should be able to search for products and identify the brick-and-mortar stores that have them in stock. They should be able to put items on hold at a store location or place an order for items that are not carried or not in stock. The functionality that the system should have is listed below:	
<ul style="list-style-type: none"> • Search through the CD Selections' inventory of products • Identify the retail stores that have the product in stock • Put a product on hold at a retail store and schedule a time to pick up the product • Place an order for products not currently in stock or not carried by CD Selections • Receive confirmation that an order can be placed and when it will be in stock 	
Business Value:	
We expect that CD Selections will increase sales by reducing lost sales due to out-of-stock or nonstocked items and by reaching out to new customers through its Internet presence. We expect the improved services will reduce customer complaints, primarily because 50 percent of all customer complaints stem from out of stocks or nonstocked items. Also, CD Selections should benefit from improved customer satisfaction and increased brand recognition due to its Internet presence.	
Conservative estimates of tangible value to the company includes:	
<ul style="list-style-type: none"> • \$750,000 in sales from new customers • \$1,875,000 in sales from existing customers • \$50,000 yearly reduction in customer service calls 	
Special Issues or Constraints:	
<ul style="list-style-type: none"> • The Marketing Department views this as a strategic system. This Internet system will add value to our current business model, and it also will serve as a proof of concept for future Internet endeavors. For example, in the future, CD Selections may want to sell products directly over the Internet. • The system should be in place for the holiday shopping season next year. 	

estimates by reducing and increasing this figure by 25 percent. This created a possible range of values from \$2,625,000 to \$4,375,000. Margaret is conservative, so she decided to include the lower number as her sales projection. See Figure 3-2 for the completed system request.

FEASIBILITY ANALYSIS

Once the need for the system and its business requirements have been defined, it is time to create a more detailed business case to better understand the opportunities and limitations associated with the proposed project. *Feasibility analysis* guides the organization in determining whether to proceed with a project. Feasibility analysis also identifies the

**YOUR
TURN**

3-2 Create a System Request

Think about your own university or college and choose an idea that could improve student satisfaction with the course enrollment process. Currently can students enroll for classes from anywhere? How long does it take? Are directions simple to follow? Is online help available?

Next, think about how technology can help support your idea. Would you need completely new technology? Can the current system be changed?

Question:

1. Create a system request that you could give to the administration that explains the sponsor, business need, business requirements, and potential value of the project. Include any constraints or issues that should be considered.

important *risks* associated with the project that must be addressed if the project is approved. As with the system request, each organization has its own process and format for the feasibility analysis, but most include three techniques: technical feasibility, economic feasibility, and organizational feasibility. The results of these techniques are combined into a *Feasibility Study* deliverable that is given to the approval committee at the end of Project Initiation. See Figure 3-3.

Although we will discuss feasibility analysis now within the context of Project Initiation, most project teams will revise their feasibility study throughout the SDLC and revisit its contents at various checkpoints during the project. If at any point the project's risks and limitations outweigh its benefits, the project team may decide to cancel the project or make necessary improvements.

Technical Feasibility: Can We Build It?

- Familiarity with Application: Less familiarity generates more risk
- Familiarity with Technology: Less familiarity generates more risk
- Project Size: Large projects have more risk
- Compatibility: The harder it is to integrate the system with the company's existing technology, the higher the risk

Economic Feasibility: Should We Build It?

- Development costs
- Annual operating costs
- Annual benefits (cost savings and revenues)
- Intangible costs and benefits

Organizational Feasibility: If We Build It, Will They Come?

- Project champion(s)
- Senior management
- Users
- Other stakeholders
- Is the project strategically aligned with the business?

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 3-3
Feasibility Analysis
Assessment Factors

Technical Feasibility

The first technique in the feasibility analysis is to assess the *technical feasibility* of the project, the extent to which the system can be successfully designed, developed, and installed by the IT group. Technical feasibility analysis is in essence a *technical risk analysis* that strives to answer the question: “*Can we build it?*¹

There are many risks that can endanger the successful completion of the project. First and foremost is the users’ and analysts’ *familiarity with the application*. When analysts are unfamiliar with the business application area, they have a greater chance of misunderstanding the users or missing opportunities for improvement. The risks increase dramatically when the users themselves are less familiar with an application, such as with the development of a system to support a new business innovation (e.g., Microsoft starting up a new Internet dating service). In general, the development of new systems is riskier than extensions to an existing system because existing systems tend to be better understood.

Familiarity with the technology is another important source of technical risk. When a system will use technology that has not been used before *within the organization*, there is a greater chance that problems will occur and delays will be incurred because of the need to learn how to use the technology. Risk increases dramatically when the technology itself is new (e.g., a new Java development toolkit).

Project size is an important consideration, whether measured as the number of people on the development team, the length of time it will take to complete the project, or the number of distinct features in the system. Larger projects present more risk, both because they are more complicated to manage and because there is a greater chance that some important system requirements will be overlooked or misunderstood. The extent to which the project is highly integrated with other systems (which is typical of large systems) can cause problems because complexity is increased when many systems must work together.

Finally, project teams need to consider the *compatibility* of the new system with the technology that already exists in the organization. Systems rarely are built in a vacuum—they are built in organizations that have numerous systems already in place. New technology and applications need to be able to integrate with the existing environment for many reasons. They may rely on data from existing systems, they may produce data that feed other applications, and they may have to use the company’s existing communications infrastructure. A new CRM system, for example, has little value if it does not use customer data found across the organization, in existing sales systems, marketing applications, and customer service systems.

The assessment of a project’s technical feasibility is not cut-and-dried because in many cases, some interpretation of the underlying conditions is needed (e.g., how large does a project need to grow before it becomes less feasible?). One approach is to compare the project under consideration with prior projects undertaken by the organization. Another option is to consult with experienced IT professionals in the organization or external IT consultants; often they will be able to judge whether a project is feasible from a technical perspective.

Economic Feasibility

The second element of a feasibility analysis is to perform an *economic feasibility* analysis (also called a *cost–benefit analysis*) that identifies the financial risk associated with the project. This attempts to answer the question: “*Should we build the system?*” Economic feasibility is determined by identifying costs and benefits associated with the system, assigning values to them, and then calculating the cash flow and return on investment for the project. The more expensive the project, the more rigorous and detailed the analysis. Figure 3-4 lists the steps to perform a cost–benefit analysis; each step will be described in the upcoming sections.

¹ We use the words “build it” in the broadest sense. Organizations can also choose to buy a commercial software package and install it, in which case, the question might be: “*Can we select the right package and successfully install it?*”

Identify Costs and Benefits The first task when developing an economic feasibility analysis is to identify the kinds of costs and benefits the system will have and list them along the left-hand column of a spreadsheet. Figure 3-5 lists examples of costs and benefits that may be included.

1. Identify Costs and Benefits	List the tangible costs and benefits for the project. Include both one-time and recurring costs.
2. Assign Values to Costs and Benefits	Work with business users and IT professionals to create numbers for each of the costs and benefits. Even intangibles should be valued if at all possible.
3. Determine Cash Flow	Project what the costs and benefits will be over a period of time, usually three to five years. Apply a growth rate to the numbers, if necessary.
4. Determine Net Present Value	Calculate what the value of future costs and benefits are if measured by today's standards. You will need to select a rate of growth to apply the NPV formula.
5. Determine Return on Investment	Calculate how much money the organization will receive in return for the investment it will make using the ROI formula.
6. Calculate Break-Even Point	Find the first year in which the system has greater benefits than costs. Apply the break-even formula using figures from that year. This will help you understand how long it will take before the system creates real value for the organization.
7. Graph Break-Even Point	Plot the yearly costs and benefits on a line graph. The point at which the lines cross is the break-even point.

FIGURE 3-4
Steps to Conduct Economic Feasibility

Development Costs	Operational Costs
Development Team Salaries	Software Upgrades
Consultant Fees	Software Licensing Fees
Development Training	Hardware Repairs
Hardware and Software	Hardware Upgrades
Vendor Installation	Operational Team Salaries
Office Space and Equipment	Communications Charges
Data Conversion Costs	User Training
Tangible Benefits	Intangible Benefits
Increased Sales	Increased Market Share
Reductions in Staff	Increased Brand Recognition
Reductions in Inventory	Higher Quality Products
Reductions in IT Costs	Improved Customer Service
Better Supplier Prices	Better Supplier Relations

FIGURE 3-5
Example Costs and Benefits for Economic Feasibility

The costs and benefits can be broken down into four categories (1) development costs, (2) operational costs, (3) tangible benefits, and (4) intangibles. *Development costs* are those tangible expenses that are incurred during the construction of the system, such as salaries for the project team, hardware and software expenses, consultant fees, training, and office space and equipment. Development costs are usually thought of as one-time costs. *Operational costs* are those tangible costs that are required to operate the system, such as the salaries for operations staff, software licensing fees, equipment upgrades, and communications charges. Operational costs are usually thought of as ongoing costs.

Revenues and cost savings are those *tangible benefits* that the system enables the organization to collect or *tangible expenses* that the system enables the organization to avoid. Tangible benefits may include increased sales, reductions in staff, and reductions in inventory.

Of course, a project also can affect the organization's bottom line by reaping *intangible benefits* or incurring *intangible costs*. Intangible costs and benefits are more difficult to incorporate into the economic feasibility because they are based on intuition and belief rather than "hard numbers." Nonetheless, they should be listed in the spreadsheet along with the tangible items.

Assign Values to Costs and Benefits Once the types of costs and benefits have been identified, you will need to assign specific dollar values to them. This may seem impossible—how can someone quantify costs and benefits that haven't happened yet? And how can those predictions be realistic? Although this task is very difficult, you have to do the best you can to come up with reasonable numbers for all of the costs and benefits. Only then can the approval committee make an educated decision about whether or not to move ahead with the project.

The best strategy for estimating costs and benefits is to rely on the people who have the best understanding of them. For example, costs and benefits that are related to the technology or the project itself can be provided by the company's IT group or external consultants, and business users can develop the numbers associated with the business (e.g., sales projections, order levels). You also can consider past projects, industry reports, and vendor information, although these approaches probably will be a bit less accurate. Likely, all of the estimates will be revised as the project proceeds.

What about the *intangible* costs and benefits? Sometimes, it is acceptable to list intangible benefits, such as improved customer service, without assigning a dollar value; whereas, other times estimates have to be made regarding how much an intangible benefit is "worth." We suggest

CONCEPTS

3-C Intangible Value at Carlson Hospitality

IN ACTION

I conducted a case study at Carlson Hospitality, a global leader in hospitality services, encompassing more than 1,300 hotel, resort, restaurant, and cruise ship operations in 79 countries. One of its brands, Radisson Hotels & Resorts, researched guest stay information and guest satisfaction surveys. The company was able to quantify how much of a guest's lifetime value can be attributed to his or her perception of the stay experience. As a result, Radisson knows how much of the collective future value of the enterprise is at stake given the perceived quality of stay experience. Using this model, Radisson can confidently show that a 10 percent

increase in customer satisfaction among the 10 percent of highest quality customers, will capture one-point market share for the brand. Each point in market share for the Radisson brand is worth \$20 million in additional revenue.

—Barbara Wixom

Question:

- How can a project team use this information to help determine the economic feasibility of a system?

that you quantify intangible costs or benefits if at all possible. If you do not, how will you know if they have been realized? Suppose that a system is supposed to improve customer service. This is intangible, but let's assume that the greater customer service will decrease the number of customer complaints by 10 percent each year over three years and that \$200,000 is spent on phone charges and phone operators who handle complaint calls. Suddenly we have some very tangible numbers with which to set goals and measure the original intangible benefit.

Figure 3-6 shows costs and benefits along with assigned dollar values. Notice that the customer service intangible benefit has been quantified based on fewer customer complaint phone calls. The intangible benefit of being able to offer services that competitors currently offer was not quantified, but it was listed so that the approval committee will consider the benefit when assessing the system's economic feasibility.

Determine Cash Flow A formal cost–benefit analysis usually contains costs and benefits over a selected number of years (usually three to five years) to show cash flow over time (see Figure 3-7). When using this *cash flow method*, the years are listed across the top of the spreadsheet to represent the time period for analysis, and numeric values are entered in the appropriate cells within the spreadsheet's body. Sometimes fixed amounts are entered into the columns. For example, Figure 3-7 lists the same amount for customer complaint calls and inventory costs for all five years. Usually amounts are augmented by some rate of growth to adjust for inflation or business

Benefits^a	
Increased sales	500,000
Improved customer service ^b	70,000
Reduced inventory costs	68,000
Total benefits	638,000
 Development costs	
2 servers @ \$125,000	250,000
Printer	100,000
Software licenses	34,825
Server software	10,945
Development labor	1,236,525
Total development costs	1,632,295
 Operational costs	
Hardware	54,000
Software	20,000
Operational labor	111,788
Total operational costs	185,788
 Total costs	1,818,083

^a An important yet intangible benefit will be the ability to offer services that our competitors currently offer.

^b Customer service numbers have been based on reduced costs for customer complaint phone calls.

FIGURE 3-6
Assign Values to Costs
and Benefits

	2003	2004	2005	2006	2007	Total
Increased sales	500,000	530,000	561,800	595,508	631,238	
Reduction in customer complaint calls	70,000	70,000	70,000	70,000	70,000	
Reduced inventory costs	68,000	68,000	68,000	68,000	68,000	
Total Benefits:	638,000	668,000	699,800	733,508	769,238	
PV of Benefits:	619,417	629,654	640,416	651,712	663,552	3,204,752
PV of All Benefits:	619,417	1,249,072	1,889,488	2,541,200	3,204,752	
2 Servers @ \$125,000	250,000	0	0	0	0	
Printer	100,000	0	0	0	0	
Software licenses	34,825	0	0	0	0	
Server software	10,945	0	0	0	0	
Development labor	1,236,525	0	0	0	0	
Total Development Costs:	1,632,295	0	0	0	0	
Hardware	54,000	81,261	81,261	81,261	81,261	
Software	20,000	20,000	20,000	20,000	20,000	
Operational labor	111,788	116,260	120,910	125,746	130,776	
Total Operational Costs:	185,788	217,521	222,171	227,007	232,037	
Total Costs:	1,818,083	217,521	222,171	227,007	232,037	
PV of Costs:	1,765,129	205,034	203,318	201,693	200,157	2,575,331
PV of All Costs:	1,765,129	1,970,163	2,173,481	2,375,174	2,575,331	
Total Project Benefits—Costs:	(1,180,083)	450,479	477,629	506,501	537,201	
Yearly NPV:	(1,145,712)	424,620	437,098	450,019	463,395	629,421
Cumulative NPV:	(1,145,712)	(721,091)	(283,993)	166,026	629,421	
Return on Investment:	24.44%	(629,421/2,575,331)				
Break-even Point:	3.63 years	(break-even occurs in year 4; [450,019 – 166,026] / 450,019 = 0.63)				
Intangible Benefits:	This service is currently provided by competitors Improved customer satisfaction					

FIGURE 3-7 Cost-Benefit Analysis

improvements, as shown by the 6 percent increase that is added to the sales numbers in the sample spreadsheet. Finally, totals are added to determine what the overall benefits will be, and the higher the overall total, the more feasible the solution is in terms of its economic feasibility.

Determine Net Present Value and Return on Investment There are several problems with the cash flow method because it does not consider the time value of money (i.e., a dollar today is not worth a dollar tomorrow), and it does not show the overall “bang for the buck” that the organization is receiving from its investment. Therefore, some project teams add additional calculations to the spreadsheet to provide the approval committee with a more accurate picture of the project’s worth.

Net present value (NPV) is used to compare the present value of future cash flows with the investment outlay required to implement the project. Consider the table in Figure 3-8 that shows the future worth of a dollar investment today, given different numbers of years

Number of years	6%	10%	15%
1	0.943	0.909	0.870
2	0.890	0.826	0.756
3	0.840	0.751	0.572
4	0.792	0.683	0.497

This table shows how much a dollar today is worth 1–4 years from now in today's terms across different interest rates.

FIGURE 3-8
The Value of a Future
Dollar Today

and different rates of change. If you have a friend who owes you a dollar today, but instead she gives you a dollar three years from now—you've been had! Given a 10 percent increase in value, you'll be receiving the equivalent of 75 cents in today's terms.

NPV can be calculated in many different ways, some of which are extremely complex. See Figure 3-9 for a basic calculation that can be used in your cash flow analysis to get more relevant values. In Figure 3-7, the present value of the costs and benefits are calculated first (i.e., they are shown at a discounted rate). Then NPV is calculated, and it shows the discounted rate of the combined costs and benefits.

The *return on investment (ROI)* is a calculation that is listed somewhere on the spreadsheet that measures the amount of money an organization receives in return for the money it spends—a high ROI results when benefits far outweigh costs. ROI is determined by taking the total benefits less the costs of the system and dividing that number by the total costs of the system (see Figure 3-9). ROI can be determined per year, or for the entire project over a period of time. One drawback of ROI is that it only considers the end points of the investment, not the cash flow in between, so it should not be used as the sole indicator of a project's worth. Find the ROI figure on the spreadsheet in Figure 3-7.

Calculation	Definition	Formula
Present Value (PV)	The amount of an investment today compared to that same amount in the future, taking into account inflation and time.	$\frac{\text{Amount}}{(1 + \text{interest rate})^n}$ <p>n = number of years in future</p>
Net Present Value (NPV)	The present value of benefit less the present value of costs.	PV Benefits – PV Costs
Return on Investment (ROI)	The amount of revenues or cost savings results from a given investment.	$\frac{\text{Total benefits} - \text{Total costs}}{\text{Total costs}}$
Break-Even Point	The point in time at which the costs of the project equal the value it has delivered.	$\frac{\text{Yearly NPV}^* - \text{Cumulative NPV}}{\text{Yearly NPV}^*}$

*Use the Yearly NPV amount from the first year in which the project has a positive cash flow.
Add the above amount to the year in which the project has a positive cash flow.

FIGURE 3-9 Financial Calculations Used For Cost-Benefit Analysis

CONCEPTS

3-D Return on Investment

IN ACTION

In 1996, IDC conducted a study of sixty-two companies that had implemented data warehousing. They found that the implementations generated an average three-year return on investment of 401 percent. The interesting part is that while forty-five companies reported results between 3 percent and 1,838 percent, the range varied from—1,857 percent to as high as 16,000 percent!

Questions:

1. What kinds of reasons would you give for the varying degrees of return on investment?
2. How would you interpret these results if you were a manager thinking about investing in data warehousing for your company?

Determine Break-Even Point If the project team needs to perform a rigorous cost–benefit analysis, they may need to include information about the length of time before the project will break-even, or when the returns will match the amount invested in the project. The greater the time it takes to break-even, the riskier the project. The *break-even point* is determined by looking at the cash flow over time and identifying the year in which the benefits are larger than the costs (see Figure 3-7). Then, the yearly and cumulative NPV for that year are divided by the yearly NPV to determine how far into the year the break-even will occur. See Figure 3-9 for the break-even calculation.

The break-even point also can be depicted graphically as shown in Figure 3-10. The cumulative present value of the costs and benefits for each year are plotted on a line graph, and the point at which the lines cross is the break-even point.

Alternatives to Traditional Cost–Benefit Analysis There have been concerns raised as to the appropriateness of using traditional cost–benefit analysis using NPV and ROI to determine economic feasibility of an IT project. One of the major problems of using traditional cost–benefit analysis to determine the economic feasibility of an IT investment is that traditional cost–benefit analysis is based on the assumption that the investor must either invest now or not invest at all. However, in most IT investment decisions, the decision to invest is not a now-or-never decision. In most situations, an information system is already in place. As such, the decision to replace or upgrade the current information system can usually be delayed. There have been different proposals

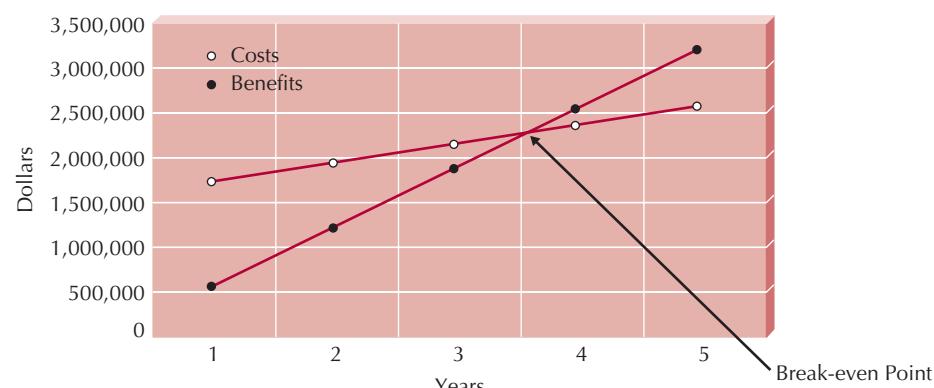


FIGURE 3-10
Break-even Graph

made to overcome some of the weaknesses in traditional cost-benefit analysis.² In this section, we describe the primary alternative that has been proposed for object-oriented systems: *option pricing models (OPMs)*.³

At this point in time, OPMs have had limited use in economic feasibility analysis for IT investment decisions in industry. In fact, there is some controversy as to whether an instrument created for a traded asset (stock) can be used in evaluating IT investment opportunities. However, the preliminary research results demonstrate that their use in IT investment evaluations may be warranted. OPMs have shown promise in evaluating the potential future value of an investment in IT. In many cases in which traditional cost–benefit analysis of investments in IT have predicted that the investment would be a failure, OPMs have shown that they may indeed be feasible.

With object-oriented systems, where classes are designed not only for the current application, but also for use in future development efforts, an investment in developing a class or a set of classes may pay “dividends” well beyond the original system development effort. Furthermore, with the iterative and incremental development emphasis in the object-oriented systems development approaches, such as those described in Chapter 2, an object-oriented project can be viewed as a sequence of smaller projects. As such, you might treat investments in an object-oriented project much as you would an investment in a call option in finance. A *call option* is essentially a contract that gives the right to purchase an amount of stock for a given price for a specified period of time to the purchaser of the call option. However, a call option does not create an obligation to buy the stock.

Treating an IT investment as a call option allows management, at a relevant point in the future, to determine whether additional investment into the evolving system is reasonable. This gives management the flexibility to determine the economic feasibility of a project to decide whether to continue with the development as planned, to abandon the project, to expand the scope of the project, to defer future development, or to shrink the current development effort. In many ways, treating IT investments as call options simply allows management to delay investment decisions until more information is available.

Once the decision is made to invest (i.e., the call option is exercised) the decision is considered irreversible. The idea of irreversible decisions is one of the fundamental assumptions on which OPMs are based. This assumption fits quite well with modern object-oriented system development approaches, in which, once an iteration has begun, an increment is completed before another investment decision is made.

Researchers have studied many different OPMs in terms of their applicability to IT investment.⁴ However, all OPMs share a common thread: both the direct benefit of the proposed

² See, for example, Q. Hu, R. Plant, and D. Hertz, “Software Cost Estimation Using Economic Production Models,” *Journal of MIS* 15(1) (Summer 1998), pp. 143–163; and G. Ooi and C. Soh, “Developing an Activity-based Approach for System Development and Implementation,” *ACM Data Base for Advances in Information Systems* 34(3) (Summer 2003), pp. 54–71.

³ For more information regarding the use of option pricing models in evaluating economic feasibility of information systems, see M. Benaroch and R. Kauffman, “A Case for Using Real Options Pricing Analysis to Evaluate Information Technology Project Investments,” *Information Systems Research* 10(1) (March 1999), pp. 70–86; M. Benaroch and R. Kauffman, “Justifying Electronic Banking Network Expansion Using Real Options Analysis,” *MIS Quarterly* 24(2) (June 2000), pp. 197–225; Q. Dai, R. Kauffman, and S. March, “Analyzing Investments in Object-Oriented Middleware,” *Ninth Workshop on Information Technologies and Systems* (December 1999), pp. 45–50; A. Kambil, J. Henderson, and H. Mohsenzadeh, “Strategic Management of Information Technology Investments: An Options Perspective,” in R.D. Banker, R.J. Kauffman, and M.A. Mahmood (eds.), *Strategic Information Technology Management: Perspectives on Organizational Growth and Competitive Advantage*, Idea Group, 1993; A. Taudes, “Software Growth Options,” *Journal of MIS* 15(1), (Summer 1998), pp. 165–185; A. Taudes, M. Feurstein, and A. Mild, “Options Analysis of Software Platform Decisions: A Case Study,” *MIS Quarterly* 24(2) (June 2000), pp. 227–243.

⁴ Two of the more important OPMs used for evaluating IT investments are the binomial OPM and the Black-Scholes OPM. For more information on these models see J.C. Hull, *Options, Futures, and Other Derivative Securities* (Englewood Cliffs, NJ: Prentice-Hall, 1993).

project and the indirect value (option value) must be computed to determine economic feasibility of an IT investment using an OPM. The direct benefit can be computed using the traditional NPV while the value of the option can be computed using one of the OPMs in the literature. Given that the minimum expected value of an option is always zero, the minimum estimated value for investing using an OPM will be the same as the value given by the traditional approach. However, when the expected value of the option (e.g., future iterations or projects) exceeds zero, an OPM will give an estimated value greater than the traditional approach. The actual calculation of the value of an option is quite complex and, as such, is beyond the scope of this book. However, given how well the OPMs fit the object-oriented systems development approaches, it seems reasonable that OPMs should be considered as alternatives to evaluating IT investments in object-oriented systems.

Organizational Feasibility

The final technique used for feasibility analysis is to assess the *organizational feasibility* of the system, how well the system ultimately will be accepted by its users and incorporated into the ongoing operations of the organization. There are many organizational factors that can have an impact on the project, and seasoned developers know that organizational feasibility can be the most difficult feasibility dimension to assess. In essence, an organizational feasibility analysis attempts to answer the question: “If we build it, will they come?”

One way to assess the organizational feasibility of the project is to understand how well the goals of the project align with business objectives. *Strategic alignment* is the fit between the project and business strategy—the greater the alignment, the less risky the project will be from an organizational feasibility perspective. For example, if the Marketing Department has decided to become more customer focused, then a CRM project that produces integrated customer information would have strong strategic alignment with Marketing’s goal. Many IT projects fail when the IT department initiates them because there is little or no alignment with business unit or organizational strategies.

A second way to assess organizational feasibility is to conduct a *stakeholder analysis*.⁵ A *stakeholder* is a person, group, or organization that can affect (or will be affected by) a new system. In general, the most important stakeholders in the introduction of a new system are the project champion, system users, and organizational management (see Figure 3-11), but systems sometimes affect other stakeholders as well. For example, the IS department can be a stakeholder of a system because IS jobs or roles may be changed significantly after its implementation. One key stakeholder outside of the champion, users, and management in Microsoft’s project that embedded Internet Explorer as a standard part of Windows was the U.S. Department of Justice.

The *champion* is a high-level non-IS executive who is usually but not always the person who created the system request. The champion supports the project by providing time, resources (e.g., money), and political support within the organization by communicating the importance of the system to other organizational decision makers. More than one champion is preferable because if the champion leaves the organization, the support could leave as well.

While champions provide day-to-day support for the system, *organizational management* also needs to support the project. Such management support conveys to the rest of the organization the belief that the system will make a valuable contribution and that necessary resources will be made available. Ideally, management should encourage people in the organization to use the system and to accept the many changes that the system will likely create.

⁵ A good book on stakeholder analysis that presents a series of stakeholder analysis techniques is R. O. Mason and I.I. Mitroff, *Challenging Strategic Planning Assumptions: Theory, Cases, and Techniques*, New York, NY: John Wiley & Sons, 1981.

	Role	Techniques for improvement
Champion	A champion: <ul style="list-style-type: none"> • Initiates the project • Promotes the project • Allocates his or her time to project • Provides resources 	<ul style="list-style-type: none"> • Make a presentation about the objectives of the project and the proposed benefits to those executives who will benefit directly from the system • Create a prototype of the system to demonstrate its potential value
Organizational Management	Organizational managers: <ul style="list-style-type: none"> • Know about the project • Budget enough money for the project • Encourage users to accept and use the system 	<ul style="list-style-type: none"> • Make a presentation to management about the objectives of the project and the proposed benefits • Market the benefits of the system using memos and organizational newsletters • Encourage the champion to talk about the project with his or her peers
System Users	Users: <ul style="list-style-type: none"> • Make decisions that influence the project • Perform hands-on activities for the project • Ultimately determine whether the project is successful by using or not using the system 	<ul style="list-style-type: none"> • Assign users official roles on the project team • Assign users specific tasks to perform with clear deadlines • Ask for feedback from users regularly (e.g., at weekly meetings)

FIGURE 3-11 Some Important Stakeholders for Organizational Feasibility

A third important set of stakeholders is the *system users* who ultimately will use the system once it has been installed in the organization. Too often, the project team meets with users at the beginning of a project and then disappears until after the system is created. In this situation, rarely does the final product meet the expectations and needs of those who are supposed to use it because needs change and users become savvier as the project progresses. User participation should be promoted throughout the development process to make sure that the final system will be accepted and used by getting users actively involved in the development of the system (e.g., performing tasks, providing feedback, and making decisions).

The final feasibility study helps organizations make wiser investments regarding IS because it forces project teams to consider technical, economic, and organizational factors that can affect their projects. It protects IT professionals from criticism by keeping the business units educated about decisions and positioned as the leaders in the decision-making process. Remember—the feasibility study should be revised several times during the project at points where the project team makes critical decisions about the system (e.g., before the design begins). It can be used to support and explain the critical choices that are made throughout the SDLC.

Applying the Concepts at CD Selections

The steering committee met and placed the Internet Order project high on its list of projects. A senior systems analyst, Alec Adams, was assigned to help Margaret conduct a feasibility analysis because of his familiarity with CD Selections' sales and distribution systems. He also was an avid user of the Web and had been offering suggestions for the improvement of CD Selections' Web site.

Alec and Margaret worked closely together over the next few weeks on the feasibility analysis. Figure 3-12 presents the executive summary page of the feasibility analysis; the report itself was about ten pages long, and it provided additional detail and supporting documentation.

As shown in Figure 3-12, the project is somewhat risky from a technical perspective. CD Selections has minimal experience with the proposed application and the technology

Internet Order Feasibility Analysis Executive Summary

Margaret Mooney and Alec Adams created the following feasibility analysis for the CD Selections Internet Order System Project. The System Proposal is attached, along with the detailed feasibility study. The highlights of the feasibility analysis are:

Technical Feasibility

The Internet Order System is feasible technically, although there is some risk.

CD Selections' risk regarding familiarity with Internet order applications is high

- The Marketing Department has little experience with Internet-based marketing and sales.
- The IT Department has strong knowledge of the company's existing order systems; however, it has not worked with Web-enabled order systems.
- Hundreds of retailers that have Internet Order applications exist in the marketplace.

CD Selections' risk regarding familiarity with the technology is medium

- The IT Department has relied on external consultants and an Information Service Provider to develop its existing Web environment.
- The IT Department has gradually learned about Web systems by maintaining the current Web site.
- Development tools and products for commercial Web application development are available in the marketplace, although the IT department has little experience with them.
- Consultants are readily available to provide help in this area.

The project size is considered medium risk

- The project team likely will include less than ten people.
- Business user involvement will be required.
- The project timeframe cannot exceed a year because of the Christmas holiday season implementation deadline, and it should be much shorter.

The compatibility with CD Selections' existing technical infrastructure should be good

- The current Order System is a client-server system built using open standards. An interface with the Web should be possible.
- Retail stores already place and maintain orders electronically.
- An Internet infrastructure already is in place at retail stores and at the corporate headquarters.
- The ISP should be able to scale their services to include a new Order System.

Economic Feasibility

A cost-benefit analysis was performed; see attached spreadsheet for details. A conservative approach shows that the Internet Order System has a good chance of adding to the bottom line of the company significantly.

ROI over 3 years: 229 percent

Total benefit after three years: \$3.5 million (adjusted for present value)

Break-even occurs: after 1.7 years

Intangible Costs and Benefits

- Improved customer satisfaction
- Greater brand recognition

Organizational Feasibility

From an organizational perspective, this project has low risk. The objective of the system, which is to increase sales, is aligned well with the senior management's goal of increasing sales for the company. The move to the Internet also aligns with Marketing's goal to become more savvy in Internet marketing and sales.

The project has a project champion, Margaret Mooney, Vice President of Marketing. Margaret is well positioned to sponsor this project and to educate the rest of the senior management team when necessary. To date, much of senior management is aware of and supports the initiative.

The users of the system, Internet consumers, are expected to appreciate the benefits of CD Selections' Web presence. And, management in the retail stores should be willing to accept the system, given the possibility of increased sales at the store level.

Additional Comments:

- The Marketing Department views this as a strategic system. This Internet system will add value to our current business model, and it also will serve as a proof of concept for future Internet endeavors.
- We should consider hiring a consultant with expertise in similar applications to assist with the project.
- We will need to hire new staff to operate the new system, from both the technical and business operations aspects.

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 3-12 Feasibility Analysis for CD Selections

because the ISP had been managing most of the Web site technology to date. One solution may be to hire a consultant with e-commerce experience to work with the IT department and to offer guidance. Further, the new system would have to exchange order information with the company's brick-and-mortar order system. Currently individual retail stores submit orders electronically, so receiving orders and exchanging information with the Internet systems should be possible.

The economic feasibility analysis includes refined assumptions that Margaret made in the system request. Figure 3-13 shows the summary spreadsheet that led to the conclusions on the feasibility analysis. Development costs are expected to be about \$250,000. This is a very rough estimate, as Alec has had to make some assumptions about the amount of time it will take to design and program the system. These estimates will be revised after a detailed workplan has been developed and as the project proceeds.⁶ Traditionally, operating costs include the costs of the computer operations. In this case, CD Selections has had to include the costs of business staff, because they are creating a new business unit, resulting in a total of about \$450,000 each year. Margaret and Alec have decided to use a conservative estimate for revenues although they note the potential for higher returns. This shows that the project can still add significant business value, even if the underlying assumptions prove to be overly optimistic. The spreadsheet was projected over three years, and the ROI and break-even point were included.

The organizational feasibility is presented in Figure 3-12. There is a strong champion, well placed in the organization to support the project. The project originated in the business or functional side of the company, not the IS department, and Margaret has carefully built up support for the project among the senior management team.

This is an unusual system in that the ultimate end users are the consumers external to CD Selections. Margaret and Alec have not done any specific market research to see how well potential customers will react to the CD Selections system, so this is a risk.

An additional stakeholder in the project is the management team responsible for the operations of the traditional stores, and the store managers. They should be quite supportive given the added service that they now can offer. Margaret and Alec need to make sure that they are included in the development of the system so that they can appropriately incorporate it into their business processes.

YOUR TURN

3-3 Create a Feasibility Analysis

Think about the idea that you developed in "Your Turn 3-2" to improve your university or college course enrollment.

Questions:

1. List three things that influence the technical feasibility of the system.

2. List three things that influence the economic feasibility of the system.
3. List three things that influence the organizational feasibility of the system.
4. How can you learn more about the issues that affect the three kinds of feasibility?

⁶ Some of the salary information may seem high to you. Most companies use a "full cost" model for estimating salary cost in which all benefits (e.g., health insurance, retirement, payroll taxes) are included in salaries when estimating costs.

	2003	2004	2005	Total
Increased sales from new customers	0	750,000	772,500	
Increased sales from existing customers	0	1,875,000	1,931,250	
Reduction in customer complaint calls	0	50,000	50,000	
Total Benefits:	<u>0</u>	<u>2,675,000</u>	<u>2,753,750</u>	
PV of Benefits:	<u>0</u>	<u>2,521,444</u>	<u>2,520,071</u>	<u>5,041,515</u>
PV of All Benefits:	<u>0</u>	<u>2,521,444</u>	<u>5,041,515</u>	
Labor: Analysis and Design	42,000	0	0	
Labor: Implementation	120,000	0	0	
Consultant Fees	50,000	0	0	
Training	5,000	0	0	
Office Space and Equipment	2,000	0	0	
Software	10,000	0	0	
Hardware	25,000	0	0	
Total Development Costs:	254,000	0	0	
Labor: Webmaster	85,000	87,550	90,177	
Labor: Network Technician	60,000	61,800	63,654	
Labor: Computer Operations	50,000	51,500	53,045	
Labor: Business Manager	60,000	61,800	63,654	
Labor: Assistant Manager	45,000	46,350	47,741	
Labor: 3 Staff	90,000	92,700	95,481	
Software upgrades	1,000	1,000	1,000	
Software licenses	3,000	1,000	1,000	
Hardware upgrades	5,000	3,000	3,000	
User training	2,000	1,000	1,000	
Communications charges	20,000	20,000	20,000	
Marketing expenses	25,000	25,000	25,000	
Total Operational Costs:	446,000	452,700	464,751	
Total Costs:	<u>700,000</u>	<u>452,700</u>	<u>464,751</u>	
PV of Costs:	<u>679,612</u>	<u>426,713</u>	<u>425,313</u>	<u>1,531,638</u>
PV of all Costs:	<u>679,612</u>	<u>1,106,325</u>	<u>1,531,638</u>	
Total Project Costs Less Benefits:	(700,000)	2,222,300	2,288,999	
Yearly NPV:	(679,612)	2,094,731	2,094,758	3,509,878
Cumulative NPV:	(679,612)	1,415,119	3,509,878	
Return on Investment:	229.16%	(3,509,878/1,531,638)		
Break-even Point:	1.32 years	(break-even occurs in year 2; [2,094,731 – 1,415,119] / 2,094,731 = 0.32)		
Intangible Benefits:	Greater brand recognition Improved customer satisfaction			

FIGURE 3-13 Economic Feasibility Analysis for CD Selections

PROJECT SELECTION

Once the feasibility analysis has been completed, it is submitted back to the approval committee along with a revised system request. The committee then decides whether to approve the project, decline the project, or table it until additional information is available. At the project level, the committee considers the value of the project by examining the business need (found in the system request) and the risks of building the system (presented in the feasibility analysis).

Before approving the project, however, the committee also considers the project from an organizational perspective; it has to keep in mind the company's entire portfolio of projects. This way of managing projects is called *portfolio management*. Portfolio management takes into consideration the different kinds of projects that exist in an organization—large and small, high risk and low risk, strategic and tactical (see Figure 3-14 for the different ways of classifying projects). A good project portfolio will have the most appropriate mix of projects for the organization's needs. The committee acts as portfolio manager with the goal of maximizing the cost/benefit performance and other important factors of the projects in their portfolio. For example, a organization may want to keep high-risk projects to less than 20 percent of its total project portfolio.

The approval committee must be selective about where to allocate resources because the organization has limited funds. This involves *trade-offs* in which the organization must give up something in return for something else to keep its portfolio well balanced. If there are three potentially high-payoff projects, yet all have very high risk, then maybe only one of the projects will be selected. Also, there are times when a system at the project level makes good business sense, but it does not at the organization level. Thus, a project may show a very strong ROI and support important business needs for a part of the company; however, it is not selected. This could happen for many reasons—because there is no money in the budget for another system, the organization is about to go through some kind of change (e.g., a merger, an implementation of a company-wide system like an ERP system), projects that meet the same business requirements already are underway, or the system does not align well with current or future corporate strategy.

Applying the Concepts at CD Selections

The approval committee met and reviewed the Internet Order System project along with two other projects—one that called for the implementation of corporate Intranet and

Size	What is the size? How many people are needed to work on the project?
Cost	How much will the project cost the organization?
Purpose	What is the purpose of the project? Is it meant to improve the technical infrastructure? Support a current business strategy? Improve operations? Demonstrate a new innovation?
Length	How long will the project take before completion? How much time will go by before value is delivered to the business?
Risk	How likely is it that the project will succeed or fail?
Scope	How much of the organization is affected by the system? A department? A division? The entire corporation?
Return on investment	How much money does the organization expect to receive in return for the amount the project costs?

FIGURE 3-14
Ways to Classify Projects

another that proposed in-store kiosks that would provide customers with information about the CDs that the store carried. Unfortunately, the budget would only allow for one project to be approved, so the committee carefully examined the costs, expected benefits, risks, and strategic alignment of all three projects. Currently, a primary focus of upper management is increasing sales in the retail stores, and the Internet system and kiosk project best aligned with that goal. Given that both projects had equal risk, but that the Internet Order project expected a much greater return, the committee decided to fund the Internet Order System.

SUMMARY

Project Initiation

Project initiation is the point at which an organization creates and assesses the original goals and expectations for a new system. The first step in the process is to identify the business value for the system by developing a system request that provides basic information

YOUR

TURN

3-4 To Select or Not To Select

It seems hard to believe that an approval committee would not select a project that meets real business needs, has a high potential ROI, and has a positive feasibility analysis.

Think of a company you have worked for or know about. Describe a scenario in which a project may be very attractive at the project level, but not at the organization level.

CONCEPTS

IN ACTION

3-E Interview with Carl Wilson, CIO, Marriott Corporation

At Marriott, we don't have IT projects—we have business initiatives and strategies that are enabled by IT. As a result, the only time a traditional "IT project" occurs is when we have an infrastructure upgrade that will lower costs or leverage better functioning technology. In this case, IT has to make a business case for the upgrade and prove its value to the company.

The way IT is involved in business projects in the organization is twofold. First, senior IT positions are filled by people with good business understanding. Second, these people are placed on key business committees and forums where the real business happens, such as finding ways to satisfy guests. Because IT has a seat at the table, we are able to spot opportunities to support business strategy. We look for ways in which IT can enable or better support business initiatives as they arise.

Therefore, business projects are proposed, and IT is one component of them. These projects are then evaluated

the same as any other business proposal, such as a new resort—by examining the return on investment and other financial measures.

At the organizational level, I think of projects as must-do's, should-do's, and nice-to-do's. The "must do's" are required to achieve core business strategy, such as guest preference. The "should do's" help grow the business and enhance the functionality of the enterprise. These can be somewhat untested, but good drivers of growth. The "nice-to-do's" are more experimental and look farther out into the future.

The organization's project portfolio should have a mix of all three kinds of projects, with a much greater proportion devoted to the "must-do's."

—Carl Wilson

CONCEPTS

IN ACTION

3-F A Project That Does Not Get Selected

Hygeia Travel Health is a Toronto-based health insurance company whose clients are the insurers of foreign tourists to the United States and Canada. Its project selection process is relatively straightforward. The project evaluation committee, consisting of six senior executives, splits into two groups. One group includes the CIO, along with the heads of operations and research and development, and it analyzes the costs of every project. The other group consists of the two chief marketing officers and the head of business development, and they analyze the expected benefits. The groups are permanent, and to stay objective, they don't discuss a project until both sides have evaluated it. The results are then shared, both on a spreadsheet and in conversation. Projects are then approved, passed over, or tabled for future consideration.

Last year, the marketing department proposed purchasing a claims database filled with detailed information on the costs of treating different conditions at different facilities. Hygeia was to use this information to estimate how much money insurance providers were likely to owe on a given claim if a patient was treated at a certain hospital as opposed to any other. For example, a 45-year-old man suffering a heart attack may accrue \$5,000 in treatment costs at hospital A, but only \$4,000 at hospital B. This information

would allow Hygeia to recommend the cheaper hospital to its customer. That would save the customer money and help differentiate Hygeia from its competitors.

The benefits team used the same three-meeting process to discuss all the possible benefits of implementing the claims database. Members of the team talked to customers and made a projection using Hygeia's past experience and expectations about future business trends. The verdict: The benefits team projected a revenue increase of \$210,000. Client retention would rise by 2 percent. And overall, profits would increase by 0.25 percent.

The costs team, meanwhile, came up with large estimates: \$250,000 annually to purchase the database and an additional \$71,000 worth of internal time to make the information usable. Put it all together and it was a financial loss of \$111,000 in the first year.

The project still could have been good for marketing—maybe even good enough to make the loss acceptable. But some of Hygeia's clients were also in the claims information business and therefore potential competitors. This, combined with the financial loss, was enough to make the company reject the project.

Source: "Two Teams Are Better Than One" *CIO Magazine*, July 15, 2001 by Ben Worthen.

YOUR
TURN

3-5 Project Selection

In April 1999, one of Capital Blue Cross's healthcare insurance plans had been in the field for three years but hadn't performed as well as expected. The ratio of premiums to claims payments wasn't meeting historic norms. In order to revamp the product features or pricing to boost performance, the company needed to understand why it was underperforming. The stakeholders came to the discussion already knowing they needed better extraction and analysis of usage data in order to understand product shortcomings and recommend improvements.

After listening to input from the user teams, the stakeholders proposed three options. One was to persevere with the current manual method of pulling data from flat files via ad hoc reports and retyping it into spreadsheets.

The second option was to write a program to dynamically mine the needed data from Capital's customer information control system (CICS). While the system was processing claims, for instance, the program would pull

out up-to-the-minute data at a given point in time for users to analyze.

The third alternative was to develop a decision-support system to allow users to make relational queries from a data mart containing a replication of the relevant claims and customer data.

Each of these alternatives was evaluated on cost, benefits, risks, and intangibles.

Question:

1. What are three costs, benefits, risks, and intangibles associated with each project?
2. Based on your answer to question 1, which project would you choose?

Source: "Capital Blue Cross," *CIO Magazine*, February 15, 2000, by Richard Pastore.

about the proposed system. Next the analysts perform a feasibility analysis to determine the technical, economic, and organizational feasibility of the system, and if appropriate, the system is approved, and the development project begins.

System Request

The business value for an information system is identified and then described using a system request. This form contains the project's sponsor, business need, business requirements, and business value of the information system, along with any other issues or constraints that are important to the project. The document is submitted to an approval committee who determines whether the project would be a wise investment of the organization's time and resources.

Feasibility Analysis

A feasibility analysis is then used to provide more detail about the risks associated with the proposed system, and it includes technical, economic, and organizational feasibilities. The technical feasibility focuses on whether the system *can* be built by examining the risks associated with the users' and analysts' familiarity with the application, familiarity with the technology, and project size. The economic feasibility addresses whether the system *should* be built. It includes a cost–benefit analysis of development costs, operational costs, tangible benefits, and intangible costs and benefits. Finally, the organizational feasibility assesses how well the system will be accepted by its users and incorporated into the ongoing operations of the organization. The strategic alignment of the project and a stakeholder analysis can be used to assess this feasibility dimension.

Project Selection

Once the feasibility analysis has been completed, it is submitted back to the approval committee along with a revised system request. The committee then decides whether to approve the project, decline the project, or table it until additional information is available. The project selection process takes into account all of the projects in the organization using portfolio management. The approval committee weighs many factors and makes trade-offs before a project is selected.

KEY TERMS

Approval committee	Feasibility study	Return on/investment (ROI)
Break-even point	First mover	Risk
Business need	Functionality	Special issues
Business requirement	Intangible benefits	Stakeholder
Business value	Intangible costs	Stakeholder analysis
Call option	Intangible value	Strategic alignment
Cash flow method	Net present value (NPV)	System request
Champion	Operational cost	System users
Compatibility	Option pricing models (OPMs)	Tangible benefits
Cost–benefit analysis	Organizational feasibility	Tangible expenses
Development cost	Organizational management	Tangible value
Economic feasibility	Portfolio management	Technical feasibility
Emerging technology	Project	Technical risk analysis
Familiarity with the application	Project initiation	Trade-offs
Familiarity with the technology	Project size	
Feasibility analysis	Project sponsor	

QUESTIONS

1. Give three examples of business needs for a system.
2. What is the purpose of an approval committee? Who is usually on this committee?
3. Why should the system request be created by a businessperson as opposed to an IS professional?
4. What is the difference between intangible value and tangible value? Give three examples of each.
5. What are the purposes of the system request and the feasibility analysis? How are they used in the project selection process?
6. Describe two special issues that may be important to list on a system request.
7. Describe the three techniques for feasibility analysis.
8. What factors are used to determine project size?
9. Describe a “risky” project in terms of technical feasibility.
10. Describe a project that would not be considered “risky.”
11. List two intangible benefits. Describe how these benefits can be quantified.
12. List two tangible benefits and two operational costs for a system. How would you determine the values that should be assigned to each item?
13. Explain the net present value and return on investment for a cost–benefit analysis. Why would these calculations be used?
14. What is the break-even point for the project? How is it calculated?
15. What is stakeholder analysis? Discuss three stakeholders that would be relevant for most projects.

EXERCISES

- A. Locate a news article in an IT trade magazine (e.g., *Computerworld*) about an organization that is implementing a new computer system. Describe the tangible and intangible value that the organization likely will realize from the new system.
- B. Car dealers have realized how profitable it can be to sell automobiles using the Web. Pretend you work for a local car dealership that is part of a large chain such as CarMax. Create a system request you might use to develop a Web-based sales system. Remember to list special issues that are relevant to the project.
- C. Suppose that you are interested in buying yourself a new computer. Create a cost–benefit analysis that illustrates the return on investment that you would receive from making this purchase. Computer-related Web sites (e.g., Dell Computers, Compaq Computers) should have real tangible costs that you can include in your analysis. Project your numbers out to include a three-year period of time and provide the net present value of the final total.
- D. Consider the Amazon.com Web site. The management of the company decided to extend their Web-based system to include products other than books (e.g., wine, specialty gifts). How would you have assessed the feasibility of this venture when the idea first came up? How “risky” would you have considered the project that implemented this idea? Why?
- E. Interview someone who works in a large organization and ask him or her to describe the approval process that exists for approving new development projects. What do they think about the process? What are the problems? What are the benefits?
- F. Reread the “Your Turn 3-1” box (Identify Tangible and Intangible Value). Create a list of the stakeholders that should be considered in a stakeholder analysis of this project.

MINICASES

1. The Amberssen Specialty Company is a chain of twelve retail stores that sell a variety of imported gift items, gourmet chocolates, cheeses, and wines in the Toronto area. Amberssen has an IS staff of three people who have created a simple but effective information system

of networked point-of-sale registers at the stores, and a centralized accounting system at the company headquarters. Harry Hilman, the head of Amberssen’s IS group, has just received the following memo from Bill Amberssen, Sales Director (and son of Amberssen’s founder).

Harry—it's time Amberssen Specialty launched itself on the Internet. Many of our competitors are already there, selling to customers without the expense of a retail storefront, and we should be there too. I project that we could double or triple our annual revenues by selling our products on the Internet. I'd like to have this ready by Thanksgiving, in time for the prime holiday gift-shopping season. Bill

After pondering this memo for several days, Harry scheduled a meeting with Bill so that he could clarify Bill's vision of this venture. Using the standard content of a system request as your guide, prepare a list of questions that Harry needs to have answered about this project.

2. The Decker Company maintains a fleet of ten service trucks and crews that provide a variety of plumbing, heating, and cooling repair services to residential customers. Currently, it takes on average about six hours before a service team responds to a service request. Each truck and crew averages twelve service calls per week, and the average revenue earned per service call is \$150. Each truck is in service fifty weeks per year. Due to the difficulty in scheduling and routing, there is considerable slack time for each truck and crew during a typical week.

In an effort to more efficiently schedule the trucks and crews and improve their productivity, Decker management is evaluating the purchase of a prewritten routing and scheduling software package. The benefits of the system will include reduced response time to service requests and more productive service teams, but management is having trouble quantifying these benefits.

One approach is to make an estimate of how much service response time will decrease with the new system, which then can be used to project the increase in the number of service calls made each week. For example, if the system permits the average service response time to fall to four hours, management believes that each truck will be able to make sixteen service calls per week on average—an increase of four calls per week. With each truck making four additional calls per week and the average revenue per call at \$150, the revenue increase per truck per week is \$600 ($4 \times \150). With ten trucks in service fifty weeks per year, the average annual revenue increase will be \$300,000 ($\$600 \times 10 \times 50$).

Decker Company management is unsure whether the new system will enable response time to fall to four hours on average, or will be some other number. Therefore, management has developed the following range of outcomes that may be possible outcomes of the new system, along with probability estimates of each outcome occurring.

New Response Time	# Calls/Truck/Week	Likelihood
2 hours	20	20%
3 hours	18	30%
4 hours	16	50%

Given these figures, prepare a spreadsheet model that computes the expected value of the annual revenues to be produced by this new system.

CHAPTER 4

PROJECT MANAGEMENT

This chapter describes the important steps of project management, which begins in the Planning Phase and continues throughout the systems development life cycle (SDLC). First, the project manager estimates the size of the project and identifies the tasks that need to be performed. Next, he or she staffs the project and puts several activities in place to help coordinate project activities. These steps produce important project management deliverables, including the workplan, staffing plan, and standards list.

OBJECTIVES

- Become familiar with estimation.
- Be able to create a project workplan.
- Understand why project teams use timeboxing.
- Become familiar with how to staff a project.
- Understand how computer-aided software engineering, standards, and documentation improve the efficiency of a project.
- Understand how to reduce risk on a project.

CHAPTER OUTLINE

Introduction	Staffing the Project
Identifying Project Size	Staffing Plan
Function Point Approach	Motivation
Creating and Managing the Workplan	Handling Conflict
Identifying Tasks	Coordinating Project Activities
The Project Workplan	CASE Tools
Gantt Chart	Standards
PERT Chart	Documentation
Refining Estimates	Managing Risk
Scope Management	Applying the Concepts at CD Selections
Timeboxing	Staffing the Project
Evolutionary Work Breakdown	Coordinating Project Activities
Structures and Iterative Workplans	Summary

INTRODUCTION

Think about major projects that occur in people's lives, such as throwing a big party like a wedding or graduation celebration. Months are spent in advance identifying and performing all of the tasks that need to get done, such as sending out invitations and selecting a menu, and time and money are carefully allocated among them. Along the way, decisions are recorded, problems are addressed, and changes are made. The increasing popularity of the party planner, a person whose sole job is to coordinate a party, suggests how tough this job can be. In the end, the success of any party has a lot to do with the effort that went into planning along the way. System development projects can be much more complicated than the projects we encounter in our personal lives—usually, more people are involved (e.g., the organization), the costs are higher, and more tasks need to be completed. Therefore, you should not be surprised to learn that "party planners" exist for information system projects—they are called project managers.

Project management is the process of planning and controlling the development of a system within a specified time frame at a minimum cost with the right functionality.¹ A *project manager* has the primary responsibility for managing the hundreds of tasks and roles that need to be carefully coordinated. Nowadays, project management is an actual profession, and analysts spend years working on projects prior to tackling the management of them. In a 1999 *Computerworld* survey, more than half of 103 companies polled said they now offer formal project management training for IT project teams. There also is a variety of *project management software* available like Microsoft Project, Plan View, and PMOffice that support project management activities.

Although training and software are available to help project managers, unreasonable demands set by project sponsors and business managers can make project management very difficult. Too often, the approach of the holiday season, the chance at winning a proposal with a low bid, or a funding opportunity pressures project managers to promise systems long before they are able to deliver them. These overly optimistic timetables are thought to be one of the biggest problems that projects face; instead of pushing a project forward faster, they result in delays.

Thus, a critical success factor for project management is to start with a realistic assessment of the work that needs to be accomplished and then manage the project according to that assessment. This can be achieved by carefully following the four steps that are presented in this chapter: identifying the project size, creating and managing the workplan, staffing the project, and coordinating project activities. The project manager ultimately creates a workplan, staffing plan, and standards list, which are used and refined throughout the entire SDLC.

¹ A set of good books on project management for object-oriented projects are Grady Booch, *Object Solutions: Managing the Object-Oriented Project* (Menlo Park, CA: Addison-Wesley, 1996); Murray R. Cantor, *Object-Oriented Project Management with UML* (New York, NY: John Wiley & Sons, 1998); Alistair Cockburn, *Surviving Object-Oriented Projects: A Manager's Guide* (Reading, MA: Addison-Wesley, 1998); and Walker Royce, *Software Project Management: A Unified Framework* (Reading, MA: Addison-Wesley, 1998). Also, the Project Management Institute (www.pmi.org) and the Information Systems Special Interest Group of the Project Management Institute (www.pmi-issig.org) have valuable resources on project management in information systems.

IDENTIFYING PROJECT SIZE

The science (or art) of project management is in making *trade-offs* among three important concepts: the size of the system (in terms of what it does), the time to complete the project (when the project will be finished), and the cost of the project. Think of these three things as interdependent levers that the project manager controls throughout the SDLC. Whenever one lever is pulled, the other two levers are affected in some way. For example, if a project manager needs to readjust a deadline to an earlier date, then the only solution is to decrease the size of the system (by eliminating some of its functions) or to increase costs by adding more people or having them work overtime. Often, a project manager will have to work with the project sponsor to change the goals of the project, such as developing a system with less functionality or extending the deadline for the final system, so that the project has reasonable goals that can be met.

Therefore, in the beginning of the project, the manager needs to estimate each of these levers and then continuously assess how to roll out the project in a way that meets the organization's needs. *Estimation*² is the process of assigning projected values for time and effort, and it can be performed manually or with the help of an estimation software package like Costar or Construx—there are over fifty available on the market. The estimates developed at the start of a project are usually based on a range of possible values (e.g., the design phase will take three to four months) and gradually become more specific as the project moves forward (e.g., the design phase will be completed on March 22).

The numbers used to calculate these estimates can come from several sources. They can be provided with the methodology that is used, taken from projects with similar tasks and technologies, or provided by experienced developers. Generally speaking, the numbers should be conservative. A good practice is to keep track of the actual time and effort values during the SDLC so that numbers can be refined along the way, and the next project

CONCEPTS

IN ACTION

4-A Trade-offs

I was once on a project to develop a system that should have taken a year to build. Instead, the business need demanded that the system be ready within five months—impossible!

On the first day of the project, the project manager drew a triangle on a white board to illustrate some trade-offs that he expected to occur over the course of the project. The corners of the triangle were labeled Functionality, Time, and Money. The manager explained, "We have too little time. We have an unlimited budget. We will not be measured by the bells and whistles that this system contains. So over the next several weeks, I want you as developers to keep this triangle in mind and do everything it takes to meet this five-month deadline."

At the end of the five months, the project was delivered on time; however, the project was incredibly over

budget, and the final product was "thrown away" after it was used because it was unfit for regular usage. Remarkably, the business users felt that the project was very successful because it met the very specific business needs for which it was built. They believed that the trade-offs that were made were worthwhile.

—Barbara Wixom

Questions:

1. What are the risks in stressing only one corner of the triangle?
2. How would you have managed this project? Can you think of another approach that might have been more effective?

² A good book for further reading on software estimation is that by Capers Jones, *Estimating Software Costs*, New York: McGraw-Hill, 1989.

can benefit from real data. One of the greatest strengths of systems consulting firms is the past experience that they offer to a project; they have estimates and methodologies that have been developed and honed over time and applied to hundreds of projects.

There are two basic ways to estimate the time required to build a system. The simplest method uses the amount of time spent in the planning phase to predict the time required for the entire project. The idea is that a simple project will require little planning and a complex project will require more planning, so using the amount of time spent in the planning phase is a reasonable way to estimate overall project time requirements.

With this approach, you take the time spent in (or estimated for) the planning phase and use industry standard percentages (or percentages from the organization's own experiences) to calculate estimates for the other SDLC phases. Industry standards suggest that a "typical" business application system spends 15 percent of its effort in the planning phase, 20 percent in the analysis phase, 35 percent in the design phase, and 30 percent in the implementation phase. This would suggest that if a project takes four months in the planning phase, then the rest of the project likely will take a total of 22.66 person-months ($4 \div 0.15 = 22.66$). These same industry percentages are then used to estimate the amount of time in each phase (Figure 4-1). The obvious limitation of this approach is that it can be difficult to take into account the specifics of your individual project, which may be simpler or more difficult than the "typical" project.

Function Point Approach³

The second approach to estimation, sometimes called the *function point approach*, uses a more complex—and, it is hoped, more reliable—three-step process (Figure 4-2). First, the project manager estimates the size of the project in terms of the number of lines of code the new system will require. This size estimate is then converted into the amount of effort required to develop the system in terms of the number of person-months. The estimated effort is then converted into an estimated schedule time in terms of the number of months from start to finish.

Step 1: Estimate System Size The first step is to estimate the size of a project using function points, a concept developed in 1979 by Allen Albrecht of IBM. A *function point* is a measure of program size that is based on the system's number and complexity of inputs, outputs, queries, files, and program interfaces.

FIGURE 4-1
Estimating Project Time
Using the Planning
Phase Approach

	Planning	Analysis	Design	Implementation
Typical industry standards for business applications	15%	20%	35%	30%
Estimates based on actual figures for first stages of SDLC	Actual: 4 person-months	Estimated: 5.33 person-months	Estimated: 9.33 person-months	Estimated: 8 person-months
SDLC = systems development life cycle.				

³ A set of good books that focus on function points are J. Brian Dreger, *Function Point Analysis* (Englewood Cliffs, NJ: Prentice Hall, 1989) and C. R. Symons, *Software Sizing and Estimating: MK II FPA* (New York, NY, John Wiley & Sons, 1991). Additional information on function point analysis and can be found at www.ifpug.org. We introduce a variation on function points, use case points, in Chapter 6.

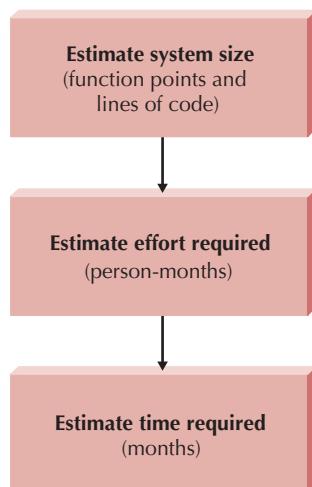


FIGURE 4-2
Estimating Project Time
Using the Function
Point Approach

To calculate the function points for a project, components are listed on a worksheet to represent the major elements of the system. For example, data-entry screens are kinds of inputs, reports are outputs, and database queries are kinds of queries (see Figure 4-3). The project manager records the total number of each component that the system will include, and then he or she breaks down the number to show the number of components that have low, medium, and high complexity. In Figure 4-3, there are nineteen outputs that need to be developed for the system, four of which have low complexity, ten that have medium complexity, and five that are very complex. After each line is filled in, a total number of points are calculated per line by multiplying each number by a complexity index. The line totals are added up to determine the *total unadjusted function points (TUFP)* for the project.

The *complexity* of the overall system is greater than the sum of its parts. Things like the familiarity of the project team with the business area and the technology that will be used to implement the project also may influence how complex a project will be. A project that is very complex for a team with little experience might have little complexity for a team with lots of experience. To create a more realistic size for the project, a number of additional system factors, such as end-user efficiency, reusability, and data communications are assessed in terms of their effect on the project's complexity (see Figure 4-3). These assessments are totaled and placed into a formula to calculate an *adjusted project complexity (APC)* score. The TUFP value is multiplied by the APC value to determine the ultimate size of the project in terms of *total adjusted function points (TAFP)*. This number should give the project manager a reasonable idea as to how big the project will be.

Sometimes a shortcut is used to determine the complexity of the project. Instead of calculating the complexity for the 14 factors listed in Figure 4-3, project managers choose to assign an APC value that ranges from 0.65 for very simple systems to 1.00 for “normal” systems to as much as 1.35 for complex systems and multiply the value to the TUFP score. For example, a very simple system that has 200 unadjusted function points would have a size of 130 adjusted function points ($200 \times 0.65 = 130$). However, if the system with 200 unadjusted function points were very complex, its function point size would be 270 ($200 \times 1.35 = 270$).

In the Planning Phase, the exact nature of the system has not yet been determined, so it is impossible to know *exactly* how many inputs, outputs, and so forth will be in the system. It is up to the project manager to make an intelligent guess. Some people feel that using function points early on in a project is not practical for this reason. We believe function points can be a useful tool for understanding a project’s size at any point in the SDLC. Later in the project, once more is known about the system, the project manager will revise the estimates using this better knowledge to produce more accurate results.

Once you have estimated the number of function points, you need to convert the number of function points into the lines of code that will be required to build the system. The number of lines of code depends on the programming language you choose to use. Figure 4-4 presents a very rough conversion guide for some popular languages.

System Components:

Description	Total Number	Complexity			Total	
		Low	Medium	High		
Inputs	6	3	3	2	23	
Outputs	19	4	4	10	5	101
Queries	10	7	3	0	4	39
Files	15	0	7	15	10	150
Program Interfaces	3	1	5	0	7	25
Total Unadjusted Function Points (TUFP):					<u>338</u>	

Overall System:

Data communications	<u>3</u>
Heavy use configuration	<u>0</u>
Transaction rate	<u>0</u>
End-user efficiency	<u>0</u>
Complex processing	<u>0</u>
Installation ease	<u>0</u>
Multiple sites	<u>0</u>
Performance	<u>0</u>
Distributed functions	<u>2</u>
Online data entry	<u>2</u>
Online update	<u>0</u>
Reusability	<u>0</u>
Operational ease	<u>0</u>
Extensibility	<u>0</u>
Total Processing Complexity (PC):	<u>7</u>

TEMPLATE
can be found at
www.wiley.com/college/dennis

(0 = no effect on processing complexity; 3 = great effect on processing complexity)

Adjusted Processing Complexity (APC):

$$0.65 + (0.01 \times 7) = 0.72$$

Total Adjusted Function Points (TAFP):

$$0.72 \text{ (APC)} \times 338 \text{ (TUFP)} = 243 \text{ (TAFP)}$$

FIGURE 4-3
Function
Point-Estimation
Worksheet

CONCEPTS

IN ACTION

4-B Function Points at Nielsen

Nielsen Media used function point analysis (FPA) for an upgrade to the Global Sample Management System (GSMS) for Nielsen Media/NetRatings, which keeps track of the Internet rating sample, a group of 40,000 homes nationwide that volunteer to participate in ongoing ratings.

In late fall of 1998, Nielsen Media did a FP count based on the current GSMS. (FPA is always easier and more accurate when there is an existing system.) Nielsen Media had its counters—three quality assurance staff—do their FPA, and then input their count into KnowledgePlan, a productivity modeling tool. In early 1999, seven programmers began writing code for the system, which they were expected to complete in ten months. As November approached, the project was adding staff to try to meet the deadline. When it became evident that the deadline

would not be met, a new FP count was conducted. The GSMS had grown to 900 FPs. Besides the original 500 plus 20 percent, there were 300 FPs attributable to features and functions that had crept into the project.

How did that happen? The way it always does: The developers and users had added a button here, a new feature there, and soon the project was much larger than it was originally. But Nielsen Media had put a stake in the ground at the beginning from which they could measure growth along the way.

The best practice is to run the FPA and productivity model at the project's launch and again when there is a full list of functional requirements. Then do another analysis anytime there is a major modification in the functional definition of the project.

Source: "Ratings Game," CIO Magazine, October 2000, by Bill Roberts.

Language	Approximate Number of Lines of Code per Function Point
C	130
COBOL	110
Java	55
C++	50
Turbo Pascal	50
Visual Basic	30
PowerBuilder	15
HTML	15
Packages (e.g., Access, Excel)	10–40

HTML = hypertext mark-up language.
Source: Capers Jones, Software Productivity Research, <http://www.spr.com>

FIGURE 4-4
Converting from
Function Points to Lines
of Code

For example, the system in Figure 4-3 has 243 function points. If you were to develop the system in COBOL, it would typically require approximately 26,730 lines of code to write it. Conversely, if you were to use Visual Basic, it typically would take 7,290 lines of code. If you could develop the system using a package such as Excel or Access, it would take between 2,430 and 9,720 lines of code. There is a great range for packages, because different packages enable you to do different things, and not all systems can be built using certain packages. Sometimes you end up writing lots of extra code to do some simple function because the package does not have the capabilities you need.

There is also a very important message from the data in this figure. Since there is a direct relationship between lines of code and the amount of effort and time required to develop a system, the choice of development language has a significant impact on the time and cost of projects.

Step 2: Estimate Effort Required Once an understanding is reached about the size of the system, the next step is to estimate the effort that is required to build it. *Effort* is a function of the system size combined with production rates (how much work someone can complete in a given time). Much research has been done on software production rates. One of the most popular algorithms, the COCOMO model,⁴ was designed by Barry W. Boehm to convert a lines-of-code estimate into a person-month estimate. There are different versions of the COCOMO model that vary based on the complexity of the software, the size of the system, the experience of the developers, and the type of software you are developing (e.g., business application software such as the registration system at

YOUR TURN

4-1 Calculate System Size

Imagine that job hunting has been going so well that you need to develop a system to support your efforts. The system should allow you to input information about the companies with which you interview, the interviews and office visits that you have scheduled, and the offers that you receive. It should be able to produce reports, such as a company contact list, an interview schedule, and an office visit schedule, as well as produce thank-you letters to be brought into a word processor to customize. You also need the system to answer queries, such as the number of interviews by city and your average offer amount.

Questions:

- Determine the number of inputs, outputs, interfaces, files, and queries that this system requires. For each element, determine if the complexity is

low, medium, or high. Record this information on a worksheet similar to the one in Figure 4-3.

- Calculate the total function points for each line on your worksheet by multiplying the number of each element with the appropriate complexity score.
- Sum up the total unadjusted function points.
- Suppose the system will be built by you using Visual Basic (VB). Given your VB skills, multiply the TUFF score by the APC score that best estimates how complex the system will be for you to develop (0.65 = simple, 1 = average, 1.35 = complex), and calculate a TAFP value.
- Using the table in Figure 4-4, determine the number of lines of code that correspond to VB. Multiply this number with the TAFP to find the total lines of code that your system will require.

YOUR TURN

4-2 Calculate Effort and Schedule Time

Refer to the project size and lines of code that you calculated in "Your Turn 4-1."

Questions:

- Determine the effort of your project in person-months by effort multiplying your lines of code (in thousands) by 1.4.

- Calculate the schedule time in months for your project using the formula, $3.0 \times \text{person-months}^{1/3}$.
- Based on your numbers, how much time will it take to complete the project if you are the developer?

⁴ The original COCOMO model is presented by Barry W. Boehm in *Software Engineering Economics* (Englewood Cliffs, NJ: Prentice-Hall, 1981). Since then, much additional research has been done. The latest version of COCOMO, COCOMO II, is described in B.W. Boehm, C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, *Software Cost Estimation with COCOMO II* (Upper Saddle River, NJ: Prentice Hall PTR, 2000). For the latest updates on COCOMO, see <http://sunset.usc.edu/COCOMOII/cocomo.html>.

your university; commercial software such as Word; or system software such as Windows). For small to moderate-size business software projects (i.e., 100,000 lines of code and ten or fewer programmers), the model is quite simple:

$$\text{effort (in person-months)} = 1.4 \times \text{thousands of lines of code}$$

For example, let's suppose that we were going to develop a business software system requiring 10,000 lines of code. This project would typically take 14 person-months to complete. If the system in Figure 4-3 were developed in COBOL (which equates to 26,730 lines of code), it would require about 37.42 person-months of effort.

Step3: Estimate Time Required Once the effort is understood, the optimal schedule for the project can be estimated. Historical data or estimation software can be used as aids, or one rule of thumb is to determine schedule using the following equation:

$$\text{schedule time (months)} = 3.0 \times \text{person-months}^{1/3}$$

This equation is widely used, although the specific numbers vary (e.g., some estimators may use 3.5 or 2.5 instead of 3.0). The equation suggests that a project that has an effort of 14 person-months should be scheduled to take a little more than 7 months to complete. Continuing the Figure 4-3 example, the 37.42 person-months would require a little over 10 months. It is important to note that this estimate is for the analysis, design, and implementation phases; it does not include the planning phase.

CREATING AND MANAGING THE WORKPLAN

Once a project manager has a general idea of the size and approximate schedule for the project, he or she creates a *workplan*, which is a dynamic schedule that records and keeps track of all of the tasks that need to be accomplished over the course of the project. The workplan lists each task, along with important information about it, such as when it needs to be completed, the person assigned to do the work, and any deliverables that will result (Figure 4-5). The level of detail and the amount of information captured by the workplan depend on the needs of the project (and the detail usually increases as the project progresses). Usually, the workplan is the main component of the project management software that we mentioned earlier.

Workplan Information	Example
Name of the task	Perform economic feasibility
Start date	Jan 05, 2003
Completion date	Jan 19, 2003
Person assigned to the task	Project sponsor; Mary Smith
Deliverable(s)	Cost-benefit analysis
Completion status	Open
Priority	High
Resources that are needed	Spreadsheet software
Estimated time	16 hours
Actual time	14.5 hours

FIGURE 4-5
Workplan Information

To create a workplan, the project manager first identifies the tasks that need to be accomplished and determines how long they will take. Then the tasks are organized within a workplan and presented graphically using Gantt and PERT charts. All of these techniques help a project manager understand and manage the project's progress over time, and they are described in detail in the next sections.

Identify Tasks

The overall objectives for the system should be listed on the system request, and it is the project manager's job to identify all of the tasks that need to be accomplished to meet those objectives. This sounds like a daunting task—how can someone know everything that needs to be done to build a system that has never been built before?

One approach for identifying tasks is to get a list of tasks that has already been developed and to modify it. There are standard lists of tasks, or methodologies, that are available for use as a starting point. As we stated in Chapter 1, a *methodology* is a formalized approach to implementing the SDLC (i.e., it is a list of steps and deliverables). A project manager can take an existing methodology, select the steps and deliverables that apply to the current project, and add them to the workplan. If an existing methodology is not available within the organization, methodologies can be purchased from consultants or vendors, or books like this textbook can serve as guidance. Using an existing methodology is the most popular way to create a workplan because most organizations have a methodology that they use for projects.

If a project manager prefers to begin from scratch, he or she can use a structured, top-down approach whereby high-level tasks are first defined and then these are broken down into subtasks. For example, Figure 4-6 shows a list of high-level tasks that are needed to implement a new IT training class. Some of the main steps in the process include identifying vendors, creating and administering a survey, and building new classrooms. Each step is then broken down in turn and numbered in a hierarchical fashion. There are eight subtasks (i.e., 7.1–7.8) for creating and administering a survey, and there are three subtasks (7.2.1–7.2.3) that comprise the review initial survey task. A list of tasks hierarchically numbered in this way is called a *work breakdown structure (WBS)*, and it is the backbone of the project workplan.

The number of tasks and level of detail depend on the complexity and size of the project. The larger the project, the more important it becomes to define tasks at a low level of detail so that essential steps are not overlooked.

There are two basic approaches to organizing a work breakdown structure: by SDLC phase or by product. For example, if a firm decided that it needed to develop a Web site, the firm could create a work breakdown structure based on the SDLC phases: planning, analysis, design, and implementation. In this case, a typical task that would take place during planning would be feasibility analysis. This task would be broken down into the different types of feasibility analysis: technical, economic, and organizational. Each of these would be further broken down into a set of subtasks. Alternatively, the firm could organize the workplan along the lines of the different products to be developed. For example, in the case of a Web site, the products could include applets, application servers, database servers, the various sets of Web pages to be designed, a site map, and so on. Then these would be further decomposed into the different tasks associated with the phases of the SDLC. As described previously, Figure 4-6 depicts the tasks necessary for creating a new IT training class. Either way, once the overall structure is determined, tasks are identified and included in the work breakdown structure of the workplan. We return to the topic of work breakdown structures and their use in iterative planning later in this chapter.

FIGURE 4-6
Work Breakdown
Structure

Task Number	Task Name	Duration (in weeks)	Dependency	Status
1	Identify vendors	2		Complete
2	Review training materials	6	1	Complete
3	Compare vendors	2	2	In Progress
4	Negotiate with vendors	3	3	Open
5	Develop communications information	4	1	In Progress
6	Disseminate information	2	5	Open
7	Create and administer survey	4	6	Open
7.1	Create initial survey	1		Open
7.2	Review initial survey	1	7.1	Open
7.2.1	Review by Director of IT Training	1		Open
7.2.2	Review by Project Sponsor	1		Open
7.2.3	Review by Representative Trainee	1		Open
7.3	Pilot test initial survey	1	7.1	Open
7.4	Incorporate survey changes	1	7.2, 7.3	Open
7.5	Create distribution list	0.5		Open
7.6	Send survey to distribution list	0.5	7.4, 7.5	Open
7.7	Send follow-up message	0.5	7.6	Open
7.8	Collect completed surveys	1	7.6	Open
8	Analyze results and choose vendor	2	4, 7	Open
9	Build new classrooms	11	1	In Progress
10	Develop course options	3	8, 9	Open

The Project Workplan

The project workplan is the mechanism that is used to manage the tasks that are listed in the work breakdown structure. It is the project manager's primary tool for managing the project. Using it, the project manager can tell if the project is ahead or behind schedule, how well the project was estimated, and what changes need to be made to meet the project deadline.

Basically, the workplan is a table that lists all of the tasks in the work breakdown structure along with important task information, such as the people who are assigned to perform the tasks, the actual hours that the tasks took, and the variances between estimated and actual completion times (see Figure 4-6). At a minimum, the information should include the duration of the task, the current statuses of the tasks (i.e., open, complete), and the *task dependencies*, which occur when one task cannot be performed until another task is completed. For example, Figure 4-6 shows that incorporating changes to the survey (task 7.4) takes a week to perform, but it cannot occur until after the survey is reviewed (task 7.2) and pilot tested (task 7.3). Key *milestones*, or important dates, are also identified on the workplan. Presentations to the approval committee, the start of end-user training, a company retreat, and the due date of the system prototype are the types of milestones that may be important to track.

Gantt Chart

A *Gantt chart* is a horizontal bar chart that shows the same task information as the project workplan, but in a graphical way. Sometimes a picture really is worth a thousand words, and the Gantt chart can communicate the high-level status of a project much faster and easier

than the workplan. Creating a Gantt chart is simple and can be done using a spreadsheet package, graphics software (e.g., Microsoft VISIO), or a project management package.

First, tasks are listed as rows in the chart, and time is listed across the top in increments based on the needs of the projects (see Figure 4-7). A short project may be divided into hours or days; whereas, a medium-sized project may be represented using weeks or months. Horizontal bars are drawn to represent the duration of each task; the bar's beginning and end mark exactly when the task will begin and end. As people work on tasks, the appropriate bars are filled in proportionately to how much of the task is finished. Too many tasks on a Gantt chart can become confusing, so it's best to limit the number of tasks to around twenty to thirty. If there are more tasks, break them down into subtasks and create Gantt charts for each level of detail.

There are many things a project manager can see by looking quickly at a Gantt chart. In addition to seeing how long tasks are and how far along they are, the project manager also can tell which tasks are sequential, which tasks occur at the same time, and which tasks

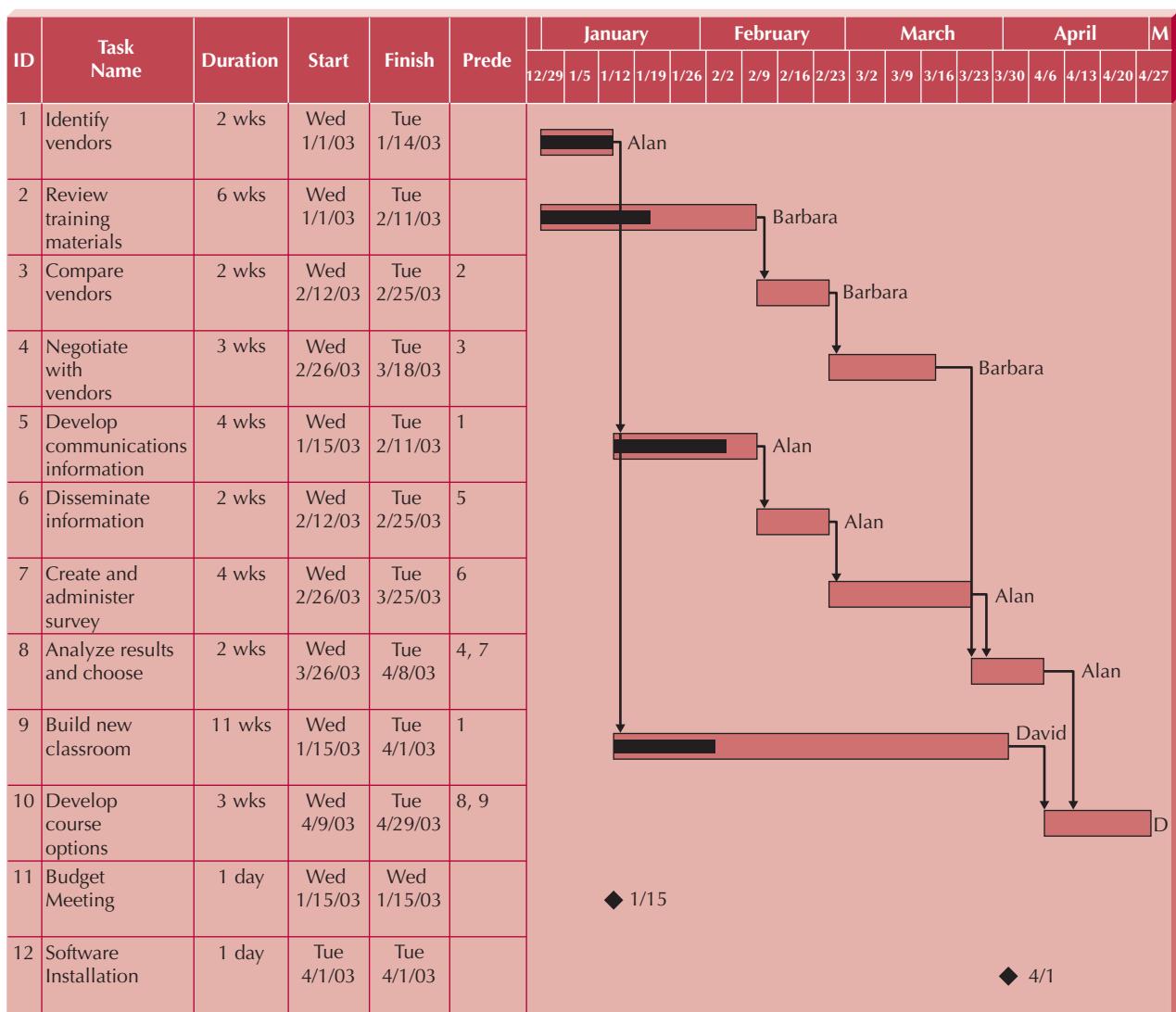


FIGURE 4-7 Gantt Chart

overlap in some way. He or she can get a quick view of tasks that are ahead of schedule and behind schedule by drawing a vertical line on today's date. If a bar is not filled in and is to the left of the line, that task is behind schedule.

There are a few special notations that can be placed on a Gantt chart. Project milestones are shown using upside-down triangles or diamonds. Arrows are drawn between the task bars to show task dependencies. Sometimes, the names of people assigned to each task are listed next to the task bars to show what human resources have been allocated to each task.

PERT Chart

A second graphical way to look at the project workplan information is the *PERT chart*, which lays out the project tasks in a flowchart (see Figure 4-8). PERT, which stands for *Program Evaluation and Review Technique*, is a network analysis technique that can be used when the individual task time estimates are fairly uncertain. Instead of simply putting a point estimate for the duration estimate, PERT uses three time estimates: optimistic, most likely, and a pessimistic. It then combines the three estimates into a single weighted average estimate using the following formula:

$$\text{PERT weighted average} = \frac{\text{optimistic estimate} + (4 \times \text{most likely estimate}) + \text{pessimistic estimate}}{6}$$

The PERT chart is drawn as a node and arc type of graph that shows the time estimates in the nodes and the task dependencies on the arcs. Each node represents an individual task, and a line connecting two nodes represents the dependency between two

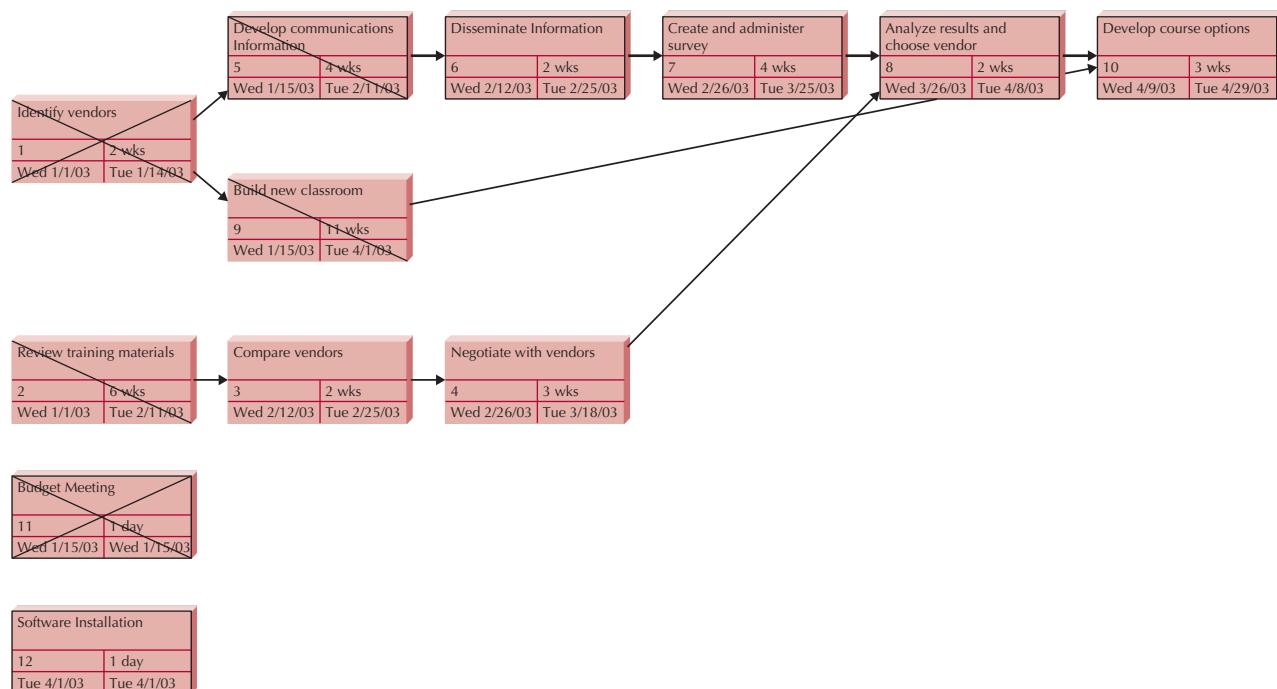


FIGURE 4-8 PERT Chart

tasks. Usually partially completed tasks are displayed with a diagonal line through the node, and completed tasks contain crossed lines.

PERT charts are the best way to communicate task dependencies because they lay out the tasks in the order in which they need to be completed. The *Critical Path Method (CPM)* simply allows the identification of the *critical path* in the network. The critical path is the longest path from the project inception to completion. The critical path shows all of the tasks that must be completed on schedule for a project as a whole to finish on schedule. If any of the tasks on the critical path takes longer than expected, the entire project will fall behind. Each task on the critical path is a *critical task*. CPM can be used with or without PERT.

The benefit of using project management software packages like Microsoft Project is that the workplan can be input once, and then the software can display the information in many different formats. You can toggle between the workplan, a Gantt chart, and a PERT chart depending on your project management needs.

Refining Estimates

The estimates that are produced during the planning phase will need to be refined as the project progresses. This does not mean that estimates were poorly done at the start of the project, but that it is virtually impossible to develop an exact assessment of the project's schedule before the analysis and design phases are conducted. A project manager should expect to be satisfied with broad ranges of estimates that become more and more specific as the project's product becomes better defined.

In many respects, estimating what an IS development project will cost, how long it will take, and what the final system will actually do follows a *hurricane model*. When storms and hurricanes first appear in the Atlantic or Pacific, forecasters watch their behavior and, on the basis of minimal information about them (but armed with lots of data on previous storms), attempt to predict when and where the storms will hit and what damage they will do when they arrive. As storms move closer to North America, forecasters refine their tracks and develop better predictions about where and when they are most likely to hit and their force when they do. The predictions become more and more accurate as the storms approach a coast, until they finally arrive.

In the Planning Phase when a system is first requested, the project sponsor and project manager attempt to predict how long the SDLC will take, how much it will cost, and what it will ultimately do when it is delivered (i.e., its functionality). However, the estimates are based on very little knowledge of the system. As the system moves into the Analysis Phase, more information is gathered, the system concept is developed, and the estimates become even more accurate and precise. As the system moves closer to completion, the accuracy and precision increase until the final system is delivered (Figure 4-9).

According to one of the leading experts in software development,⁵ a well-done project plan (prepared at the end of the planning phase) has a 100 percent margin of error for project cost and a 25 percent margin of error for schedule time. In other words, if a carefully done project plan estimates that a project will cost \$100,000 and take twenty weeks, the project will actually cost between \$0 and \$200,000 and take between fifteen and twenty-five weeks. Figure 4-10 presents typical margins of error for other stages in the project. It is important to note that these margins of error apply only to well-done plans; a plan developed without much care has a much greater margin of error.

⁵ Barry W. Boehm and colleagues, "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0," in J. D. Arthur and S. M. Henry (editors), *Annals of Software Engineering: Special Volume on Software Process and Product Measurement*, Amsterdam: J. C. Baltzer AG Science Publishers, 1995.

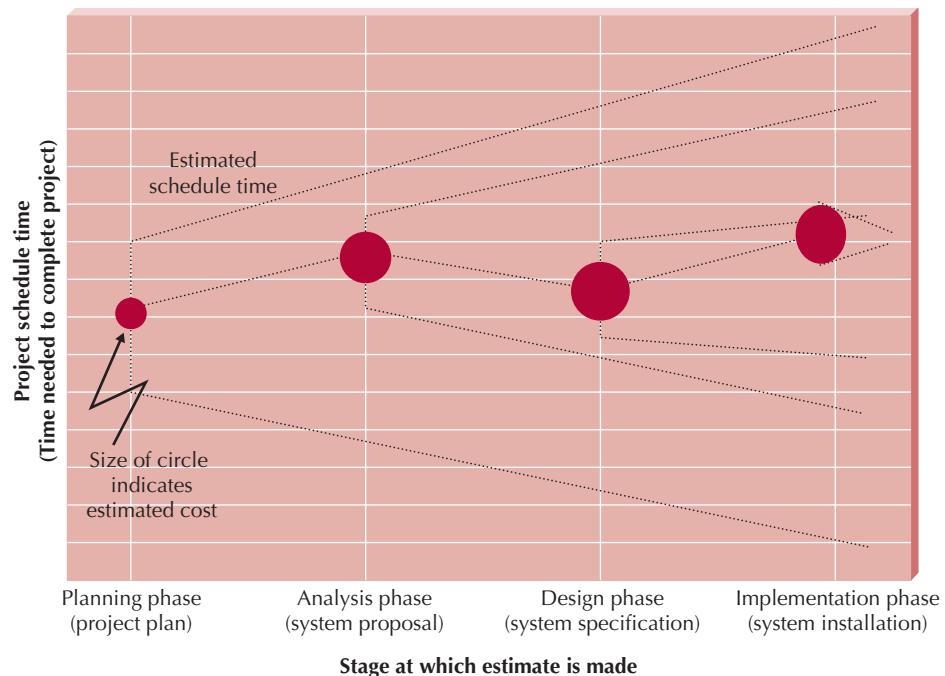


FIGURE 4-9
Hurricane Model

Typical Margins of Error for Well-Done Estimates			
Phase	Deliverable	Cost (%)	Schedule Time (%)
Planning phase	System request	400	60
	Project plan	100	25
Analysis phase	System proposal	50	15
Design phase	System specifications	25	10

Source: Barry W. Boehm and colleagues, "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0," in J. D. Arthur and S. M. Henry (editors) *Annals of Software Engineering Special Volume on Software Process and Product Measurement*, Amsterdam: J. C. Baltzer AG Science Publishers, 1995.

FIGURE 4-10
Margins of Error in Cost
and Time Estimates

What happens if you overshoot an estimate (e.g., the Analysis Phase ends up lasting two weeks longer than expected)? There are number of ways to adjust future estimates. If the project team finishes a step ahead of schedule, most project managers shift the deadlines sooner by the same amount but do not adjust the promised completion date. The challenge, however, occurs when the project team is late in meeting a scheduled date. Three possible responses to missed schedule dates are presented in Figure 4-11. We recommend that if an estimate proves too optimistic early in the project, do not expect to make up for lost time—very few projects end up doing this. Instead, change your future estimates to include an increase similar to the one that was experienced. For example, if the first phase was completed 10 percent over schedule, increase the rest of your estimates by 10 percent.

Scope Management

You may assume that your project will be safe from scheduling problems because you carefully estimated and planned your project up front. However, the most common reason for schedule and cost overruns occurs after the project is underway—*scope creep*.

Assumptions	Actions	Level of Risk
If you assume the rest of the project is simpler than the part that was late and is also simpler than believed when the original schedule estimates were made, you can make up lost time	Do not change schedule.	High risk
If you assume the rest of the project is simpler than the part that was late and is no more complex than the original estimate assumed, you can't make up the lost time, but you will not lose time on the rest of the project.	Increase the entire schedule by the total amount of time that you are behind (e.g., if you missed the scheduled date by two weeks, move the rest of the schedule dates to two weeks later). If you included padded time at the end of the project in the original schedule, you may not have to change the promised system delivery date; you'll just use up the padded time.	Moderate risk
If you assume that the rest of the project is as complex as the part that was late (your original estimates too optimistic), then all the scheduled dates in the future underestimate the real time required by the same percentage as the part that was late.	Increase the entire schedule by the percentage of weeks that you are behind (e.g., if you are two weeks late on part of the project that was supposed to take eight weeks, you need to increase all remaining time estimates by 25 percent). If this moves the new delivery date beyond what is acceptable to the project sponsor, the scope of the project must be reduced.	Low risk

FIGURE 4-11
Possible Actions When
a Schedule Date
Is Missed

Scope creep happens when new requirements are added to the project after the original project scope was defined and “frozen.” It can happen for many reasons: users may suddenly understand the potential of the new system and realize new functionality that would be useful; developers may discover interesting capabilities to which they become very attached; a senior manager may decide to let this system support a new strategy that was developed at a recent board meeting.

Unfortunately, after the project begins, it becomes increasingly difficult to address changing requirements. The ramifications of change become more extensive, the focus is removed from original goals, and there is at least some impact on cost and schedule. Therefore, the project manager plays a critical role in managing this change to keep scope creep to a minimum.

The keys are to identify the requirements as well as possible in the beginning of the project and to apply analysis techniques effectively. For example, if needs are fuzzy at the project’s onset, a combination of intensive meetings with the users and prototyping could be used so that users “experience” the requirements and better visualize how the system could support their needs. In fact, the use of meetings and prototyping has been found to reduce scope creep to less than 5 percent on a typical project.

Of course, some requirements may be missed no matter what precautions you take, but several practices can be helpful to control additions to the task list. First, the project manager should allow only absolutely necessary requirements to be added after the project begins. Even at that point, members of the project team should carefully assess the ramifications of the addition and present the assessment back to the users. For example, it may

require two more person-months of work to create a newly defined report, which would throw off the entire project deadline by several weeks. Any change that is implemented should be carefully tracked so that an audit trail exists to measure the change's impact.

Sometimes changes cannot be incorporated into the present system even though they truly would be beneficial. In this case, these additions to scope should be recorded as future enhancements to the system. The project manager can offer to provide functionality in future releases of the system, thus getting around telling someone no.

Timeboxing

Another approach to scope management is a technique called *timeboxing*. Up until now, we have described projects that are task oriented. In other words, we have described projects that have a schedule that is driven by the tasks that need to be accomplished, so the greater number of tasks and requirements, the longer the project will take. Some companies have little patience for development projects that take a long time, and these companies take a time-oriented approach that places meeting a deadline above delivering functionality.

Think about your use of word processing software. For 80 percent of the time, you probably use only 20 percent of the features, such as the spelling checker, boldfacing, and cutting and pasting. Other features, such as document merging and creation of mailing labels, may be nice to have, but they are not a part of your day-to-day needs. The same goes for other software applications; most users rely on only a small subset of their capabilities. Ironically, most developers agree that typically 75 percent of a system can be provided relatively quickly, with the remaining 25 percent of the functionality demanding most of the time.

To resolve this incongruity, a technique called timeboxing has become quite popular, especially when using rapid application development (RAD) methodologies. This technique sets a fixed deadline for a project and delivers the system by that deadline no matter what, even if functionality needs to be reduced. Timeboxing ensures that project teams don't get hung up on the final "finishing touches" that can drag out indefinitely, and it satisfies the business by providing a product within a relatively fast time frame.

There are several steps to implement timeboxing on a project (Figure 4-12). First, set the date of delivery for the proposed goals. The deadline should not be impossible to meet, so it is best to let the project team determine a realistic due date. Next, build the core of the system to be delivered; you will find that timeboxing helps create a sense of urgency and helps keep the focus on the most important features. Because the schedule is absolutely fixed, functionality that cannot be completed needs to be postponed. It helps if the team prioritizes a list of features beforehand to keep track of what functionality the users absolutely need. Quality cannot be compromised, regardless of other constraints, so it is important that the time allocated to activities is not shortened unless the requirements are changed (e.g., don't reduce the time allocated to testing without reducing features). At the end of the time period, a high-quality system is delivered; likely, future iterations will be needed to make changes and enhancements, and the timeboxing approach can be used once again.

1. Set the date for system delivery.
2. Prioritize the functionality that needs to be included in the system.
3. Build the core of the system (the functionality ranked as most important).
4. Postpone functionality that cannot be provided within the time frame.
5. Deliver the system with core functionality.
6. Repeat steps 3 through 5, to add refinements and enhancements.

FIGURE 4-12
Steps for Timeboxing

Evolutionary Work Breakdown Structures and Iterative Workplans

Since object-oriented systems approaches to systems analysis and design support incremental and iterative development, any project planning approach for object-oriented systems development requires an incremental and iterative process also. In the description of the enhanced Unified Process in Chapter 2, the development process was organized around iterations, phases, and workflows. In many ways, a workplan for an incremental and iterative development process is organized in a similar manner. For each iteration, there are different tasks that are executed on each workflow. In our minimized approach (MOOSAD), each build is analogous to the iterations of the enhanced Unified Process and each step is similar to the workflows (see Chapter 2). In this section, we describe an incremental and iterative process using evolutionary work breakdown structures for project planning that can be used with object-oriented systems development.

According to Royce,⁶ most approaches to developing conventional WBSs tend to have three underlying problems:

- 1.** *They tend to be focused on the design of the information system being developed.* As such, the creation of the WBS forces the decomposition of the system design and the tasks associated with creating the design of the system prematurely. Where the problem domain is well understood, tying the structure of the workplan to the product to be created makes sense. However, in cases where the problem domain is not well understood, the analyst must commit to the architecture of the system being developed before the requirements of the system are fully understood.
- 2.** *They tend to force too many levels of detail very early on in the SDLC for large projects or they tend to allow too few levels of detail for small projects.* Since the primary purposes of a WBS is to allow cost estimation and scheduling to take place, in conventional approaches to planning, the WBS must be done “correctly and completely” at the beginning of the SDLC. To say the least, this is a very difficult task to accomplish with any degree of validity. In such cases, it is no wonder why

CONCEPTS

4-C Timeboxing

IN ACTION

DuPont was one of the first companies to use timeboxing. The practice originated in the company's fibers division, which was moving to a highly automated manufacturing environment. It was necessary to create complex application software quickly, and DuPont recognized that it is better to get a basic version of the system working, learn from the experience of operating with it, and then design an enhanced version than it is to wait for a comprehensive system at a later date.

DuPont's experience implies that:

- The first version must be built quickly.
- The application must be built so that it can be changed and added to quickly.

DuPont stresses that the timebox methodology works well for the company and is highly practical. It has resulted in automation being introduced more rapidly and effectively. DuPont quotes large cost savings from the methodology, and variations of timebox techniques have since been used in many other corporations.

Questions:

- 1.** Why do you think DuPont saves money by using this technique?
- 2.** Are there situations in which timeboxing would not be appropriate?

Source: "Within the Timebox, Development Deadlines Really Work," PC Week, March 12, 1990, by James Martin.

⁶ Walker Royce, *Software Project Management: A Unified Framework* (Reading, MA: Addison-Wesley, 1998).

cost and schedule estimation for many information systems development projects tend to be wildly inaccurate.

3. *Since they are project specific, they are very difficult to compare across projects.* This leads to ineffective learning across the organization. Without some standard approach to create WBSs, it is difficult for project managers to learn from previous projects managed by others. This tends to encourage the “reinventing of wheels” and allows managers to make the same mistakes that previous managers have made.

Evolutionary WBSs allow the analyst to address all three of these problems by allowing the development of an *iterative workplan*. First, they are organized in a standard manner across all projects: by workflows, phases, and then tasks. This decouples the structure of an evolutionary WBS from the structure of the design of the product. This prevents prematurely committing to a specific architecture of a new system. Second, evolutionary WBSs are created in an incremental and iterative manner. The first evolutionary WBS is typically only done for the aspects of the project understood by the analyst. Later on, as the analyst understands more about the evolving development process, more details are added to the WBS. This encourages a more realistic view of both cost and schedule estimation. Third, since the structure of an evolutionary WBS is not tied to any specific project, evolutionary WBSs enable the comparison of the current project to earlier projects. This supports learning from past successes and failures.

In the case of the enhanced Unified Process, the workflows are the major points listed in the WBS. Next, each workflow is decomposed along the phases of the enhanced Unified Process. After that, each phase is decomposed along the tasks that are to be completed to create the deliverables associated with the phase. The tasks listed for each workflow is dependent upon the level of activity on the workflow during each phase (see Figure 2-9). For example, typical activities for the inception phase of the requirements workflow would be to interview stakeholders, develop vision document, and develop use cases, while there are probably no tasks associated with the inception phase of the operations and support workflow. The template for the first two levels of an evolutionary WBS for the enhanced Unified Process would look like Figure 4-13. As each iteration through the development process is completed, additional tasks are added to the WBS to reflect the current understanding of the remaining tasks to be completed (i.e., the WBS evolves along with the evolving information system). As such, the workplan is developed in an incremental and iterative manner.

Using our minimized approach (see Figure 2-10), the first evolutionary WBS would only focus on the planning step, requirements-gathering and use-case development step, and the first build. After the first build is completed, a new incremental version of the WBS would be created to take into account the information that flowed from the first build back to the requirements-determination and use-case development step, which, in turn, flowed back into the planning step. The new WBS would then address these issues and issues related to the second build. This iterative planning process would continue until the system was completed or the project was abandoned. The template for an evolutionary WBS for our minimized approach is shown in Figure 4-14.

STAFFING THE PROJECT

Staffing the project includes determining how many people should be assigned to the project, matching people's skills with the needs of the project, motivating them to meet the

project's objectives, and minimizing the conflict that will occur over time. The deliverable for this part of project management is a staffing plan, which describes the number and kinds of people who will work on the project, the overall reporting structure, and the project charter, which describes the project's objectives and rules.

Staffing Plan

The first step to staffing is determining the average number of staff needed for the project. To calculate this figure, divide the total person-months of effort by the optimal schedule.

I. Business Modeling	a. Inception	a. Inception
b. Elaboration	b. Elaboration	b. Elaboration
c. Construction	c. Construction	c. Construction
d. Transition	d. Transition	d. Transition
e. Production	e. Production	e. Production
II. Requirements	VI. Test	X. Environment
a. Inception	a. Inception	a. Inception
b. Elaboration	b. Elaboration	b. Elaboration
c. Construction	c. Construction	c. Construction
d. Transition	d. Transition	d. Transition
e. Production	e. Production	e. Production
III. Analysis	VII. Deployment	XI. Operations and Support
a. Inception	a. Inception	a. Inception
b. Elaboration	b. Elaboration	b. Elaboration
c. Construction	c. Construction	c. Construction
d. Transition	d. Transition	d. Transition
e. Production	e. Production	e. Production
IV. Design	VIII. Configuration and Change Management	XII. Infrastructure Management
a. Inception	a. Inception	a. Inception
b. Elaboration	b. Elaboration	b. Elaboration
c. Construction	c. Construction	c. Construction
d. Transition	d. Transition	d. Transition
e. Production	e. Production	e. Production
V. Implementation	IX. Project Management	

FIGURE 4-13
Evolutionary WBS Template for the Enhanced Unified Process

1. Planning	3.1.1. Analysis
1.1. First Build	3.1.2. Design
1.2. Second Build	3.1.3. Implementation
...	3.2. Second Build
1.3. Nth Build	3.2.1. Analysis
2. Requirements Gathering and Use Case Development	3.2.2. Design
2.1. First Build	3.2.3. Implementation
2.2. Second Build	...
...	3.3. Nth Build
2.3. Nth Build	3.3.1. Analysis
3. Builds	3.3.2. Design
3.1. First Build	3.3.3. Implementation
	4. Installation

FIGURE 4-14
Evolutionary WBS Template for the MOOSAD Approach

So to complete a 40 person-month project in 10 months, a team should have an average of four full-time staff members, although this may change over time as different specialists enter and leave the team (e.g., business analysts, programmers, technical writers).

Many times, the temptation is to assign more staff to a project to shorten the project's length, but this is not a wise move. Adding staff resources does not translate into increased productivity; staff size and productivity share a disproportionate relationship, mainly because a large number of staff members is more difficult to coordinate. The more a team grows, the more difficult it becomes to manage. Imagine how easy it is to work on a two-person project team: the team members share a single line of communication. But adding two people increases the number of communication lines to six, and greater increases lead to more dramatic gains in communication complexity. Figure 4-15 illustrates the impact of adding team members to a project team.

One way to reduce efficiency losses on teams is to understand the complexity that is created in numbers and to build in a *reporting structure* that tempers its effects. The rule of thumb is to keep team sizes under eight to ten people; therefore, if more people are needed, create subteams. In this way, the project manager can keep the communication effective within small teams, which in turn communicate to a contact at a higher level in the project.

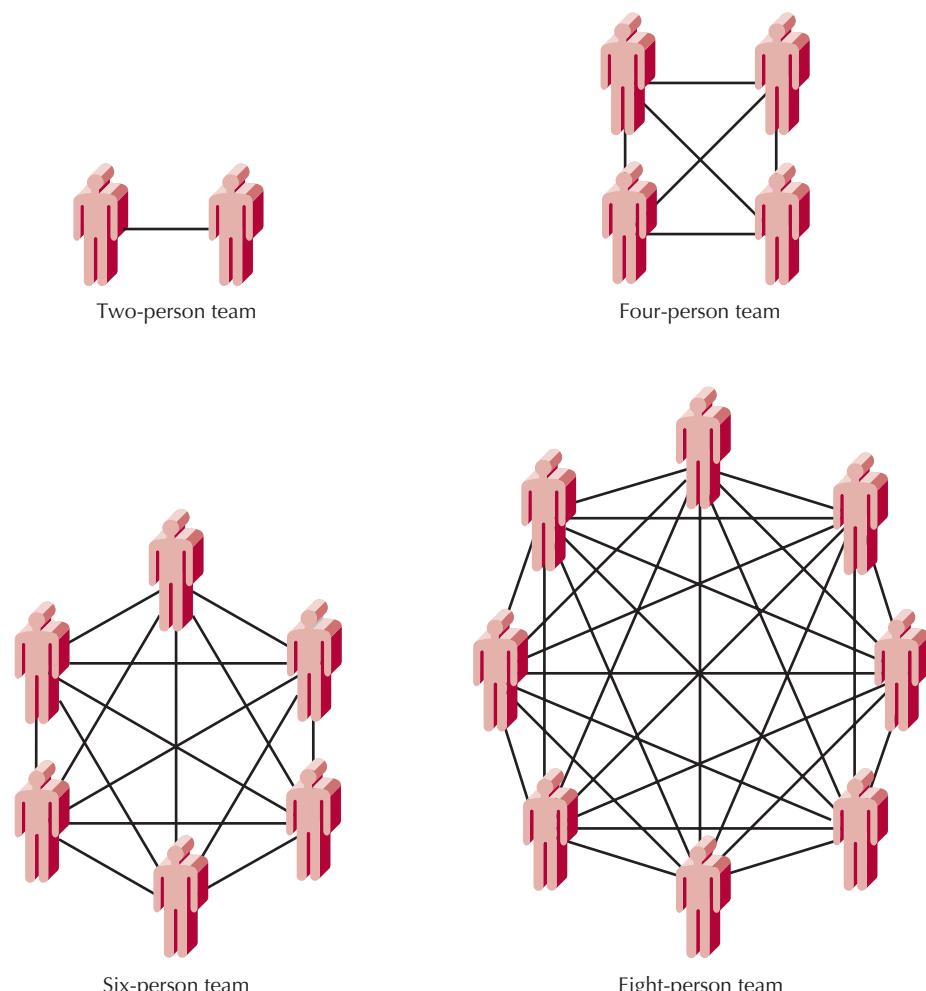


FIGURE 4-15
Increasing Complexity
with Larger Teams

After the project manager understands how many people are needed for the project, he or she creates a *staffing plan* that lists the roles that are required for the project and the proposed reporting structure for the project. Typically, a project will have one project manager who oversees the overall progress of the development effort, with the core of the team comprising the various types of analysts described in Chapter 1. A *functional lead* usually is assigned to manage a group of analysts, and *technical lead* oversees the progress of a group of programmers and more technical staff members.

There are many structures for project teams; Figure 4-16 illustrates one possible configuration of a project team. After the roles are defined and the structure is in place, the project manager needs to think about which people can fill each role. Often, one person fills more than one role on a project team.

When you make assignments, remember that people have *technical skills* and *interpersonal skills*, and both are important on a project. Technical skills are useful when working with technical tasks (e.g., programming in Java) and in trying to understand the various roles that technology plays in the particular project (e.g., how a Web server should be configured on the basis of a projected number of hits from customers).

Interpersonal skills, on the other hand, include interpersonal and communication abilities that are used when dealing with business users, senior management executives, and other members of the project team. They are particularly critical when performing the

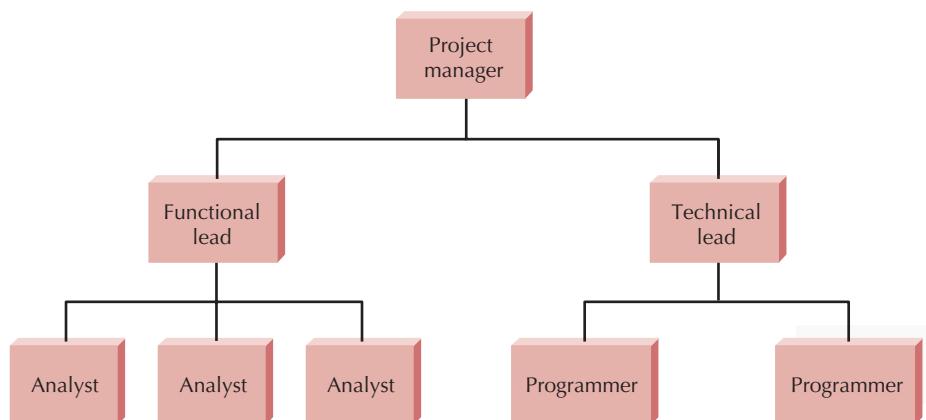


FIGURE 4-16
Possible Reporting
Structure

YOUR TURN

4-3 Staffing Plan

Now it is time to staff the project that was described in "Your Turn 4-1." On the basis of the effort required for the project ("Your Turn 4-2"), how many people will be needed on the project? Given this number, select classmates who will work with you on your project.

Questions:

- What roles will be needed to develop the project?
List them and write short descriptions for each of

these roles, almost as if you had to advertise the positions in a newspaper.

- Which roles will each classmate perform? Will some people perform multiple roles?
- What will the reporting structure be for the project?

requirements-determination activities and when addressing organizational feasibility issues. Each project will require unique technical and interpersonal skills. For example, a Web-based project may require Internet experience or Java programming knowledge, or a highly controversial project may need analysts who are particularly adept at managing political or volatile situations.

Ideally, project roles are filled with people who have the right skills for the job; however, the people who fit the roles best may not be available; they may be working on other projects, or they may not exist in the company. Therefore, assigning project team members really is a combination of finding people with the appropriate skill sets and finding people who are available. When the skills of the available project team members do not match what is actually required by the project, the project manager has several options to improve the situation. First, people can be pulled off other projects, and resources can be shuffled around. This is the most disruptive approach from the organization's perspective. Another approach is to use outside help—such as a consultant or contractor—to train team members and start them off on the right foot. Training classes are usually available for both technical and interpersonal instruction, if time is available. Mentoring may also be an option; a project team member can be sent to work on another similar project so that he or she can return with skills to apply to the current job.

Motivation

Assigning people to tasks isn't enough; project managers need to motivate the people to make the project a success. *Motivation* has been found to be the number-one influence on people's performance,⁷ but determining how to motivate the team can be quite difficult. You may think that good project managers motivate their staff by rewarding them with money and bonuses, but most project managers agree that this is the last thing that should be done. The more often you reward team members with money, the more they expect it—and most times monetary motivation won't work.

Assuming that team members are paid a fair salary, technical employees on project teams are much more motivated by recognition, achievement, the work itself, responsibility, advancement, and the chance to learn new skills.⁸ If you feel like you need to give some kind of reward for motivational purposes, try a pizza or free dinner, or even a kind letter or award. They often have much more effective results. Figure 4-17 lists some other motivational don'ts that you should avoid to ensure that motivation on the project is as high as possible.

Handling Conflict

The third component of staffing is organizing the project to minimize conflict among group members. *Group cohesiveness* (the attraction that members feel to the group and to other members) contributes more to productivity than do project members' individual capabilities or experiences.⁹ Clearly defining the roles on the project and holding team members accountable for their tasks is a good way to begin mitigating potential conflict on a project. Some project managers develop a *project charter* that lists the project's norms and ground

⁷ Barry W. Boehm, *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice Hall, 1981. One of the best books on managing project teams is by Tom DeMarco and Timothy Lister, *Peopleware: Productive Projects and Teams*, (New York, NY: Dorset House, 1987).

⁸ F. H. Hertzberg, "One More Time: How Do You Motivate Employees?" *Harvard Business Review*, 1968, January–February.

⁹ B. Lakhanpal, "Understanding the Factors Influencing the Performance of Software Development Groups: An Exploratory Group-Level Analysis," *Information and Software Technology*, 1993, 35(8):468–473.

Don'ts	Reasons
Assign unrealistic deadlines	Few people will work hard if they realize that a deadline is impossible to meet.
Ignore good efforts	People will work harder if they feel like their work is appreciated. Often, all it takes is public praise for a job well done.
Create a low-quality product	Few people can be proud of working on a project that is of low quality.
Give everyone on the project a raise	If everyone is given the same reward, then high-quality people will believe that mediocrity is rewarded—and they will resent it.
Make an important decision without the team's input	Buy-in is very important. If the project manager needs to make a decision that greatly affects the members of her team, she should involve them in the decision-making process.
Maintain poor working conditions	A project team needs a good working environment or motivation will go down the tubes. This includes lighting, desk space, technology, privacy from interruptions, and reference resources.

Source: Adapted *Rapid Development*, Redmond, WA: Microsoft Press, 1996, by from Steve McConnell.

FIGURE 4-17
Motivational Don'ts

CONCEPTS

4-D Haste Slows Microsoft

IN ACTION

In 1984, Microsoft planned for the development of Microsoft Word to take one year. At the time, this was two months less than the most optimistic estimated deadline for a project of its size. In reality, it took Microsoft five years to complete Word. Ultimately, the overly aggressive schedule for Word slowed its development for a number of reasons. The project experienced high turnover due to developer burn-out from unreasonable pressure and work hours. Code was "finalized" prematurely, and the software spent much longer in "stabilization" (i.e., fixing bugs) than was originally expected (i.e., twelve months versus three months). And, the aggressive scheduling resulted in poor

planning (the delivery date consistently was off by more than 60 percent for the first four years of the project).

Question:

1. Suppose you take over as project manager in 1986 after the previous project manager has been fired. Word is now one year overdue. Describe three things you would do on your first day on the job to improve the project.

Source: Microsoft Corporation: Office Business Unit, *Harvard Business School Case Study 9-691-033*, revised May 31, 1994, Boston: Harvard Business School, 1994, by Marco Lansiti

rules. For example, the charter may describe when the project team should be at work, when staff meetings will be held, how the group will communicate with each other, and the procedures for updating the workplan as tasks are completed. Figure 4-18 lists additional techniques that can be used at the start of a project to keep conflict to a minimum.

COORDINATING PROJECT ACTIVITIES

Like all project management responsibilities, the act of coordinating project activities continues throughout the entire project until a system is delivered to the project sponsor and end users. This step includes putting efficient development practices in place and mitigating risk. These activities occur over the course of the entire SDLC, but it is at this point in

**YOUR
TURN**

4-4 Project Charter

Get together with several of your classmates and pretend that you are all staffed on the project described in "Your Turn 4-1." Discuss what would most motivate each of you to perform well on the project. List three potential sources of conflict that could surface as you work together.

Question:

1. Develop a project charter that lists five rules that all team members will need to follow. How might these rules help avoid potential team conflict?

- Clearly define plans for the project.
- Make sure the team understands how the project is important to the organization.
- Develop detailed operating procedures and communicate these to the team members.
- Develop a project charter.
- Develop schedule commitments ahead of time.
- Forecast other priorities and their possible impact on project.

Source: H. J. Thamhain and D. L. Wilemon, "Conflict Management in Project Life Cycles," *Sloan Management Review*, Spring 1975.

FIGURE 4-18
Conflict Avoidance
Strategies

the project when the project manager needs to put them in place. Ultimately, these activities ensure that the project stays on track and that the chance of failure is kept at a minimum. The rest of this section will describe each of these activities in more detail.

CASE Tools

Computer-aided software engineering (CASE) is a category of software that automates all or part of the development process. Some CASE software packages are primarily used during the analysis phase to create integrated diagrams of the system and to store information regarding the system components (often called *upper CASE*), whereas others are design-phase tools that create the diagrams and then generate code for database tables and system functionality (often called *lower CASE*). *Integrated CASE*, or I-CASE, contains functionality found in both upper-CASE and lower-CASE tools in that it supports tasks that happen throughout the SDLC. CASE comes in a wide assortment of flavors in terms of complexity and functionality, and there are many good programs available in the marketplace, such as the Visible Analyst Workbench, Oracle Designer/2000, Rational Rose, and the Logic Works suite.

The benefits to using CASE are numerous. With CASE tools, tasks are much faster to complete and alter, development information is centralized, and information is illustrated through diagrams, which typically are easier to understand. Potentially, CASE can reduce maintenance costs, improve software quality, and enforce discipline, and some project teams even use CASE to assess the magnitude of changes to the project.

Of course, like anything else, CASE should not be considered a silver bullet for project development. The advanced CASE tools are complex applications that require significant training and experience to achieve real benefits. Often, CASE serves only as a glorified diagramming tool that supports the practices described in Chapter 6 (functional modeling), Chapter 7 (structural modeling), and Chapter 8 (behavioral modeling). Our experience has shown that CASE is a helpful way to support the communication and sharing of project diagrams and technical specifications—as long as it is used by trained developers who have applied CASE on past projects.

**YOUR
TURN****4-5 Computer-Aided Software Engineering Tool Analysis**

Select a computer-aided software engineering (CASE) tool—either one that you will use for class, a program that you own, or a tool that you can examine over the Web. Create a list of the capabilities that are offered by the CASE tool.

Question:

1. Would you classify the CASE as upper CASE, lower CASE, or integrated CASE (I-CASE)? Why?

The central component of any CASE tool is the *CASE repository*, otherwise known as the information repository or data dictionary. The CASE repository stores the diagrams and other project information, such as screen and report designs, and it keeps track of how the diagrams fit together. For example, most CASE tools will warn you if you place a field on a screen design that doesn't exist in your data model. As the project evolves, project team members perform their tasks using CASE.

Standards

Members of a project team need to work together, and most project management software and CASE tools provide access privileges to everyone working on the system. When people work together, however, things can get pretty confusing. To make matters worse, people sometimes get reassigned in the middle of a project. It is important that their project knowledge does not leave with them and that their replacements can get up to speed quickly.

One way to make certain that everyone is on the same page by performing tasks in the same way and following the same procedures is to create *standards* that the project team must follow. Standards can range from formal rules for naming files to forms that must be completed when goals are reached to programming guidelines. See Figure 4-19 for some examples of the types of standards that a project may create. When a team forms standards and then follows them, the project can be completed faster because task coordination becomes less complex.

Standards work best when they are created at the beginning of each major phase of the project and well communicated to the entire project team. As the team moves forward, new standards are added when necessary. Some standards (e.g., file-naming conventions, status reporting) are applied to the entire SDLC, whereas others (e.g., programming guidelines) are only appropriate for certain tasks.

Documentation

A final technique that project teams put in place during the planning phase is good *documentation*, which includes detailed information about the tasks of the SDLC. Often, the documentation is stored in *project binder(s)* that contain all the deliverables and all the internal communication that takes place—the history of the project.

A poor project management practice is waiting until the last minute to create documentation. This typically leads to an undocumented system that no one understands. In fact, many problems that companies had updating their systems to handle the year 2000 crisis were the result of the lack of documentation. Good project teams learn to document the system's history as it evolves while the details are still fresh in their memory.

Types of Standards	Examples
Documentation standards	The date and project name should appear as a header on all documentation. All margins should be set to 1 inch.
Coding standards	All deliverables should be added to the project binder and recorded in its table of contents.
Procedural standards	All modules of code should include a header that lists the programmer, last date of update, and a short description of the purpose of the code. Indentation should be used to indicate loops, if-then-else statements, and case statements.
Specification requirement standards	On average, every program should include one line of comments for every five lines of code.
User interface design standards	Record actual task progress in the work plan every Monday morning by 10 A.M. Report to project update meeting on Fridays at 3:30 P.M. All changes to a requirements document must be approved by the project manager.
	Name of program to be created Description of the program's purpose Special calculations that need to be computed Business rules that must be incorporated into the program Pseudocode Due date
	Labels will appear in boldface text, left-justified, and followed by a colon. The tab order of the screen will move from top left to bottom right.
	Accelerator keys will be provided for all updatable fields.

FIGURE 4-19
A Sampling of Project Standards

The first step to setting up your documentation is to get some binders and include dividers with which to separate content according to the major phases of the project. An additional divider should contain internal communication, such as the minutes from status meetings, written standards, letters to and from the business users, and a dictionary of relevant business terms. Then, as the project moves forward, place the deliverables from each task into the project binder with descriptions so that someone outside of the project will be able to understand it, and keep a table of contents up to date with the content that is added. Documentation takes time up front, but it is a good investment that will pay off in the long run.

Managing Risk

One final facet of project management is *risk management*, the process of assessing and addressing the risks that are associated with developing a project. Many things can cause risks: weak personnel, scope creep, poor design, and overly optimistic estimates. The project team must be aware of potential risks so that problems can be avoided or controlled well ahead of time.

Typically, project teams create a *risk assessment*, or a document that tracks potential risks along with an evaluation of the likelihood of the risk and its potential impact on the project (Figure 4-20). A paragraph or two is also included that explains potential ways that the risk can be addressed. There are many options: risks could be publicized, avoided, or

CONCEPTS**4-E Poor Naming Standards****IN ACTION**

I once started on a small project (four people) in which the original members of the project team had not set up any standards for naming electronic files. Two weeks into the project, I was asked to write a piece of code that would be referenced by other files that had already been written. When I finished my piece, I had to go back to the other files and make changes to reflect my new work. The only problem was that the lead programmer decided to name the files using his initials (e.g., GGI.prg, GG2.prg, GG3.prg)—and there were over 200 files! I spent two days opening every one of those files because there was no way to tell what their contents were.

Needless to say, from then on, the team created a code for file names that provided basic information regarding the file's contents and they kept a log that recorded the file name, its purpose, the date of last update, and programmer for every file on the project.

—Barbara Wixom

Question:

1. Think about a program that you have written in the past. Would another programmer be able to make changes to it easily? Why or why not?

even eliminated by dealing with its root cause. For example, imagine that a project team plans to use new technology but its members have identified a risk in the fact that its members do not have the right technical skills. They believe that tasks may take much longer to perform because of a high learning curve. One plan of attack could be to eliminate the root cause of the risk—the lack of technical experience by team members—by finding time and resources that are needed to provide proper training to the team.

Most project managers keep abreast of potential risks, even prioritizing them according to their magnitude and importance. Over time, the list of risks will change as some items are removed and others surface. The best project managers, however, work hard to keep risks from having an impact on the schedule and costs associated with the project.

CONCEPTS**4-F The Real Names of the Systems Development Life Cycle (SDLC) Phases****IN ACTION**

Dawn Adams, Senior Manager with Asymetrix Consulting, has renamed the SDLC phases:

1. Pudding (Planning)
2. Silly Putty (Analysis)
3. Concrete (Design)
4. Touch-this-and-you're-dead-sucker (Implementation)

Adams also uses icons, such as a skull and crossbones for the implementation phase. The funny labels lend a new depth of interest to a set of abstract concepts. But her names have had another benefit. "I had

one participant who adopted the names wholeheartedly," she says, "including my icons. He posted an icon on his office door for the duration of each of the phases, and he found it much easier to deal with requests for changes from the client, who could see the increasing difficulty of the changes right there on the door."

Question:

1. What would you do if your project sponsor demanded that an important change be made during the "touch-this-and-you're-dead-sucker" phase?

Source: *Learning Technology, Shorttakes*, Wednesday, August 26, 1998, 1(2).

TEMPLATE
can be found at
[www.wiley.com/
college/dennis](http://www.wiley.com/college/dennis)

FIGURE 4-20
Sample Risk
Assessment

Risk Assessment	
RISK #1:	The development of this system likely will be slowed considerably because project team members have not programmed in Java prior to this project.
Likelihood of risk:	High probability of risk.
Potential impact on the project:	This risk likely will increase the time to complete programming tasks by 50 percent.
Ways to address this risk: It is very important that time and resources are allocated to up-front training in Java for the programmers who are used for this project. Adequate training will reduce the initial learning curve for Java when programming begins. Additionally, outside Java expertise should be brought in for at least some part of the early programming tasks. This person should be used to provide experiential knowledge to the project team so that JAVA-related issues (of which novice Java programmers would be unaware) are overcome.	
RISK #2:	...

APPLYING THE CONCEPTS AT CD SELECTIONS

Alec Adams was very excited about managing the Internet Order System project at CD Selections, but he realized that his project team would have very little time to deliver at least some parts of the system because the company wanted the application developed in time for the holiday season. Therefore, he decided that the project should follow a RAD phased-development methodology, combined with the timeboxing technique. In this way, he could be sure that some version of the product would be in the hands of the users within several months, even if the completed system would be delivered at a later date.

As project manager, Alec had to estimate the project's size, effort, and schedule—some of his least favorite jobs because of how tough it is to do at the very beginning of the project. But he knew that the users would expect at least general ranges for a product delivery date. He began by attempting to estimate the number of inputs, outputs, queries, files, and program interfaces in the new system. For the Web part of the system to be used by customers, he could think of four main queries (searching by artist, by CD title, by song title, and by ad hoc criteria), three input screens (selecting a CD, entering information to put a CD on hold, and a special order screen), four output screens (the home page with general information, information about CDs, information about the customer's special order, and the hold status), three files (CD information, inventory information, and customer orders), and two program interfaces (one to the company's special order system and one that communicates hold information to the retail store systems). For the part of the system to be used by CD Selections staff (to maintain the marketing materials), he identified three additional inputs, three outputs, four queries, one file, and one program interface. He believed most of these to be of medium complexity. He entered these numbers in a worksheet (Figure 4-21).

Rather than attempt to assess the complexity of the system in detail, Alec chose to use a value of 1.20 for APC. He reasoned that the system was of medium complexity and that the project team has little to moderate experience with Internet-base systems. This produced a TAFP of about 190.

PRACTICAL**TIP**

As Seattle University's David Umphress has pointed out, watching most organizations develop systems is like watching reruns of *Gilligan's Island*. At the beginning of each episode, someone comes up with a cockamamie scheme to get off the island that seems to work for a while, but something goes wrong and the castaways find themselves right back where they started—stuck on the island. Similarly, most companies start new projects with grand ideas that seem to work, only to make a classic mistake and deliver the project behind schedule, over budget, or both. Here we summarize four classic mistakes in the planning and project management aspects of the project and discuss how to avoid them:

- 1. Overly optimistic schedule:** Wishful thinking can lead to an overly optimistic schedule that causes analysis and design to be cut short (missing key requirements) and puts intense pressure on the programmers, who produce poor code (full of bugs).
Solution: Don't inflate time estimates; instead, explicitly schedule slack time at the end of each phase to account for the variability in estimates, using the margins of error from Figure 4-10.
- 2. Failing to monitor the schedule:** If the team does not regularly report progress, no one knows if the project is on schedule.

**4-1 Avoiding Classic Planning Mistakes**

Solution: Require team members to honestly report progress (or the lack of progress) every week. There is no penalty for reporting a lack of progress, but there are immediate sanctions for a misleading report.

- 3. Failing to update the schedule:** When a part of the schedule falls behind (e.g., information gathering uses all of the slack in item 1 above plus two weeks), a project team often thinks it can make up the time later by working faster. It can't. This is an early warning that the entire schedule is too optimistic.

Solution: Immediately revise the schedule and inform the project sponsor of the new end date or use timeboxing to reduce functionality or move it into future versions.

- 4. Adding people to a late project:** When a project misses a schedule, the temptation is to add more people to speed it up. This makes the project take longer because it increases coordination problems and requires staff to take time to explain what has already been done.

Solution: Revise the schedule, use timeboxing, throw away bug-filled code, and add people only to work on an isolated part of the project.

Source: Adapted from *Rapid Development*, Redmond, WA: Microsoft Press, 1996, pp. 29–50, by Steve McConnell.

System Components:

Description	Total Number	Complexity			Total
		Low	Medium	High	
Inputs	6	0 3	4 4	2 6	28
Outputs	7	2 4	4 5	1 7	35
Queries	8	3 3	4 4	1 6	31
Files	4	0 7	4 10	0 15	40
Program Interfaces	3	0 5	2 7	1 10	24
Total Unadjusted Function Points (TUPF):					158

FIGURE 4-21
Function Points for the
Internet Order System

Adjusted Project Adjusted Processing Complexity (APC): 1.2

Total Adjusted Function Points (TAFP):

Converting function points into lines of code was challenging. The project would use a combination of C (for most programs) and HTML for the Web screens. Alec decided to assume that about 75 percent of the function points would be C and 25 percent would be HTML, which produced a total of about 19,200 lines of code [$(0.75 \times 190 \times 130) + (0.25 \times 190 \times 15)$].

Using the COCOMO formula, he found that this translated into about 27 person-months of effort (1.4×19.2). This in turn suggested a schedule time of about nine months ($3.0 \times 27^{1/3}$). After much consideration, Alec decided to pad the estimate by 10 percent (by adding one extra month).

Once the estimation was underway, Alec began to create an evolutionary work breakdown structure and iterative workplan to identify the tasks that would be needed to complete the system. He used an object-oriented systems analysis and design methodology that CD Selections had in house. His workplan is based on the steps contained in the in-house methodology (see Figure 4-22). At this point in time, Alec does not know enough to create a complete workplan. As such, he has included as much detail as he knows to be correct.

	Duration	Dependency
1. Planning		
1.1. First Build		
1.1.1. Create first version of evolutionary WBS	1 day	
1.1.2. Perform feasibility analysis	5 days	
1.1.2.1. Perform technical feasibility analysis	1 day	
1.1.2.2. Perform economic feasibility analysis	2 days	
1.1.2.3. Perform organizational feasibility analysis	2 days	
1.1.3. Identify staffing requirements for first build	0.5 days	1.1.1, 1.1.2
1.1.4. Compute first version of cost estimation	0.5 days	1.1.3
1.2. Second Build		
...		
1.3. Nth Build		
2. Requirements Determination and Use Case Development		
2.1. First Build		
2.1.1. Create requirements definition	8 days	
2.1.1.1. Determine requirements to track	1 day	
2.1.1.2. Compile requirements as they are elicited	5 days	2.1.1.1
2.1.1.3. Review requirements with sponsor	2 days	2.1.1.2
2.1.2. Elicit requirements	11.5 days	
2.1.2.1. Perform document analysis	5 days	
2.1.2.2. Conduct interviews	4.5 days	
2.1.2.2.1. Interview project sponsor	0.5 days	
2.1.2.2.2. Interview inventory system contact	0.5 days	2.1.2.2.1
2.1.2.2.3. Interview special order system contact	0.5 days	2.1.2.2.2
2.1.2.2.4. Interview ISP contact	0.5 days	2.1.2.2.3
2.1.2.2.5. Interview CD Selection Web contact	0.5 days	2.1.2.2.4
2.1.2.2.6. Interview other personnel	2 days	2.1.2.2.5
2.1.2.2.3. Observe retail store processes	2 days	
2.1.3. Analyze current system		
2.1.3.1. Draw or reverse engineer functional models		
2.1.3.2. Draw or reverse engineer structural models		
2.1.3.3. Draw or reverse engineer behavioral models		
2.1.4. Identify opportunities for improvements		
2.2. Second Build		
...		
2.3. Nth Build		

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 4-22
Iterative Workplan for
CD Selections
(Continues)

For example, Alec feels confident about the estimation of time to create the requirements definition and to elicit the requirements. However, he does not know whether the analysis models of the current system can be reverse-engineered using a CASE tool or whether they will have to be done by hand. Until this determination can be made, any estimation as to the time required would be simply a guess. As time passes, Alec expects to know much more about the development process and will add much more detail to the workplan.

Staffing the Project

Alec next turned to the task of how to staff his project. On the basis of his earlier estimates, it appeared that about three people would be needed to deliver the system by the holidays (27 person-months over 10 months of calendar time means three people, rounded up).

First, he created a list of the various roles that he needed to fill. He thought he would need several analysts to work with the analysis and design of the system as well as an infrastructure analyst to manage the integration of the Internet order system with CD

	Duration	Dependency
3. Builds		
3.1 First Build		
3.1.1. Analysis		
3.1.1.1. Create functional models		
3.1.1.2. Create structural models		
3.1.1.3. Create behavioral models		
3.1.1.4. Walkthrough analysis models		
3.1.2. Design		
3.1.2.1. Factor analysis models		
3.1.2.2. Identify partitions and collaborations		
3.1.2.3. Design problem domain classes and methods		
3.1.2.3.1. Optimize class design		
3.1.2.3.2. Restructure the design		
3.1.2.3.3. Develop contracts and method specification		
3.1.2.4. Design object persistence		
3.1.2.5. Design user interfaces		
3.1.2.6. Design physical architecture		
3.1.2.7. Walkthrough design models		
3.1.3. Implementation		
3.1.3.1. Test system implementation		
3.1.3.1.1. Perform unit tests		
3.1.3.1.2. Perform integration tests		
3.1.3.1.3. Perform system tests		
3.1.3.1.4. Perform acceptance tests		
3.1.3.2. Finalize system documentation		
3.2. Second Build		
3.2.1. Analysis		
3.2.2. Design		
3.2.3. Implementation		
...		
3.3. Nth Build		
3.3.1. Analysis		
3.3.2. Design		
3.3.3. Implementation		
4. Installation		

FIGURE 4-22
Iterative Workplan for
CD Selections
(Continued)

Selections' existing technical environment. Alec also needed people who had good programmer skills and who could be responsible for ultimately implementing the system. Ian, Anne, and K.C. are three analysts with strong technical and interpersonal skills (although Ian is less balanced, having greater technical than interpersonal abilities), and Alec believed that they were available to bring onto this project. He wasn't certain if they had experience with the actual Web technology that would be used on the project, but he decided to rely on vendor training or an external consultant to build those skills later when they were needed. Because the project was so small, Alec envisioned all of the team members reporting to him because he would be serving as the project's manager.

Alec created a staffing plan that captured this information, and he included a special incentive structure in the plan (Figure 4-23). Meeting the holiday deadline was very important to the project's success, so he decided to offer a day off to the team members who contributed to meeting that date. He hoped that this incentive would motivate the team to work very hard. Alec also planned to budget money for pizza and sodas for times when the team worked long hours.

Before he left for the day, Alec drafted a project charter, to be fine-tuned after the team got together for its *kick-off meeting* (i.e., the first time the project team gets together). The charter listed several norms that Alec wanted to put in place from the start to eliminate any misunderstanding or problems that could come up otherwise (Figure 4-24).

Coordinating Project Activities

Alec wanted the Internet Order System project to be well coordinated, so he immediately put several practices in place to support his responsibilities. First, he acquired the CASE tool used at CD Selections and set up the product so that it could be used for the analysis-phase tasks. The team members would likely start creating diagrams and defining components of the system fairly early on. He pulled out some standards that he uses on all development projects and made a note to review them with his project team at the kick-off meeting for the system. He also had his assistant set up binders for the project deliverables that would start rolling in. Already he was able to include the system request, feasibility analysis, initial workplan, staffing plan, project charter, standards list, and risk assessment.

Role	Description	Assigned To
Project manager	Oversees the project to ensure that it meets its objectives in time and within budget	Alec
Infrastructure analyst	Ensures the system conforms to infrastructure standards at CD Selections; ensures that the CD Selections infrastructure can support the new system	Ian
Systems analyst	Designs the information system—with a focus on interfaces with the distribution system	Ian
Systems analyst	Designs the information system—with a focus on the analysis models and interface design	K.C.
Systems analyst	Designs the information system—with a focus on the analysis models and system performance	Anne
Programmer	Codes system	K.C.
Programmer	Codes system	Anne
Reporting structure: All project team members will report to Alec. Special incentives: If the deadline for the project is met, all team members who contributed to this goal will receive a free day off, to be taken over the holiday season.		

FIGURE 4-23
Staffing plan for the Internet Order System

TEMPLATE
can be found at
[www.wiley.com/
college/dennis](http://www.wiley.com/college/dennis)

FIGURE 4-24
Project Charter for the Internet Order System

Project objective: The Internet Order System project team will create a working Web-based system to sell CDs to CD Selections' customers in time for the holiday season.

The Internet Order System team members will:

1. Attend a staff meeting each Friday at 2 P.M. to report on the status of assigned tasks.
2. Update the workplan with actual data each Friday by 5 P.M.
3. Discuss all problems with Alec as soon as they are detected.
4. Agree to support each other when help is needed, especially for tasks that could hold back the progress of the project.
5. Post important changes to the project on the team bulletin board as they are made.

SUMMARY

Project Management

Project management is the second major component of the planning phase of the systems development life cycle (SDLC), and it includes four steps: identifying the project size, creating and managing the workplan, staffing the project, and coordinating project activities. Project management is important in ensuring that a system is delivered on time, within budget, and with the desired functionality.

Identifying Project Size

The project manager estimates the amount of time and effort that will be needed to complete the project. First, the size is estimated by relying on past experiences or industry standards or by calculating the function points, a measure of program size based on the number and complexity of inputs, outputs, queries, files, and program interfaces. Next, the project manager calculates the effort for the project, which is a function of size and production rates. Algorithms like the COCOMO model can be used to determine the effort value. Third, the optimal schedule for the project is estimated.

Creating and Managing the Workplan

Once a project manager has a general idea of the size and approximate schedule for the project, he or she creates a workplan, which is a dynamic schedule that records and keeps track of all of the tasks that need to be accomplished over the course of the project. To create a workplan, the project manager first identifies the work breakdown structure, or the tasks that need to be accomplished, and then he or she determines how long the tasks will take. Important information about each task is entered into a workplan.

The workplan information can be presented graphically using Gantt and PERT charts. In the Gantt chart, horizontal bars are drawn to represent the duration of each task, and as people work on tasks, the appropriate bars are filled in proportionately to how much of the task is finished. PERT charts are the best way to communicate task dependencies because they lay out the tasks as a flowchart in the order in which they need to be completed. The longest path from the project inception to completion is referred to as the critical path.

Estimating what an IS development project will cost, how long it will take, and what the final system will actually do follows a hurricane model. The estimates become more accurate as the project progresses. One threat to the reliability of estimates is scope creep, which occurs when new requirements are added to the project after the original project scope was defined and "frozen." If the final schedule will not deliver the system in a timely fashion, timeboxing can be used. Timeboxing sets a fixed deadline for a project and delivers the system by that deadline no matter what, even if functionality must be reduced.

Evolutionary work breakdown structures and iterative workplans better fit the typical methodologies associated with object-oriented systems development. They allow the project manager to provide more realistic estimates for each iteration or build of a system. Furthermore, they allow the workplan to be decoupled from the architecture of the system, thus allowing projects to be comparable. By supporting comparability among projects, evolutionary WBSs enable organizational learning to take place.

Staffing the Project

Staffing involves determining how many people should be assigned to the project, assigning project roles to team members, developing a reporting structure for the team, and matching people's skills with the needs of the project. Staffing also includes motivating the team to meet the project's objectives and minimizing conflict among team members. Both motivation and cohesiveness have been found to greatly influence performance of team members in project situations. Team members are motivated most by such nonmonetary things as recognition, achievement, and the work itself. Conflict can be minimized by clearly defining the roles on a project and holding team members accountable for their tasks. Some managers create a project charter that lists the project's norms and ground rules.

Coordinating Project Activities

Coordinating project activities includes putting efficient development practices in place and mitigating risk, and these activities occur over the course of the entire SDLC. Three techniques are available to help coordinate activities on a project: computer-aided software engineering (CASE), standards, and documentation. CASE is a category of software that automates all or part of the development process; standards are formal rules or guidelines that project teams must follow during the project; and documentation includes detailed information about the tasks of the SDLC. Often, documentation is stored in project binder(s) that contain all the deliverables and all the internal communication that takes place—the history of the project. A risk assessment is used to mitigate risk because it identifies potential risks and evaluates the likelihood of risk and its potential impact on the project.

KEY TERMS

Adjusted project complexity (APC)	Group cohesiveness	Project manager
Complexity	Hurricane model	Reporting structure
Computer-aided software engineering (CASE)	Integrated CASE	Risk assessment
CASE repository	Iterative workplan	Risk management
COCOMO model	Interpersonal skills	Scope creep
Critical path	Kick-off meeting	Staffing plan
Critical path method (CPM)	Lower CASE	Standards
Critical task	Methodology	Task dependency
Documentation	Milestone	Technical lead
Effort	Motivation	Technical skills
Estimation	PERT chart	Timeboxing
Evolutionary WBS	Program evaluation and review technique	Total adjusted function points (T AFP)
Function point	Project binder	Total unadjusted function points (TU FP)
Function point approach	Project charter	Trade-offs
Functional lead	Project management	Upper CASE
Gantt chart	Project management software	Work breakdown structure
		Workplan

QUESTIONS

1. Why do many projects end up having unreasonable deadlines? How should a project manager react to unreasonable demands?
2. What are the trade-offs that project managers must manage?
3. What are two basic ways to estimate the size of a project?
4. What is a function point, and how is it used?
5. Describe the three steps of the function point approach.
6. What is the formula for calculating the effort for a project?
7. Name two ways to identify the tasks that need to be accomplished over the course of a project.
8. What is the difference between a methodology and a workplan? How are the two terms related?
9. Compare and contrast the Gantt chart and the PERT chart.
10. Describe the hurricane model.
11. What is scope creep, and how can it be managed?
12. What is timeboxing, and why is it used?
13. What are the problems associated with conventional WBSs?
14. What is an evolutionary WBS? How do they address the problems associated with conventional WBSs?
15. What is an iterative workplan?
16. Describe the differences between a technical lead and a functional lead. How are they similar?
17. Describe three technical skills and three interpersonal skills that would be very important to have on any project.
18. What are the best ways to motivate a team? What are the worst ways?
19. List three techniques to reduce conflict.
20. What is the difference between upper CASE (computer-aided software engineering) and lower CASE?
21. Describe three types of standards, and provide examples of each.
22. What belongs in the project binder? How is the project binder organized?
23. Create a list of potential risks that could affect the outcome of a project.
24. Some companies hire consulting firms to develop the initial project plans and manage the project, but use their own analysts and programmers to develop the system. Why do you think some companies do this?

EXERCISES

- A. Visit a project management Web site, such as the Project Management Institute (www.pmi.org). Most have links to project management software products, white papers, and research. Examine some of the links for project management to better understand a variety of Internet sites that contain information related to this chapter.
- B. Select a specific project management topic like computer-aided software engineering (CASE), project management software, or timeboxing and search for information on that topic using the Web. The URL listed in question A or any search engine (e.g., Google) can provide a starting point for your efforts.
- C. Pretend that the Career Services office at your university wants to develop a system that collects student résumés and makes them available to students and recruiters over the Web. Students should be able to input their résumé information into a standard résumé template. The information then is presented in a résumé format, and it also is placed in a database that can be queried using an online search form. You have been placed in charge of the project. Develop a plan for estimating the project. How long do you think it would take for you and three other students to complete the project? Provide support for the schedule that you propose.
- D. Refer to the situation in question C. You have been told that recruiting season begins a month from today and that the new system must be used. How would you approach this situation? Describe what you can do as the project manager to make sure that your team does not burn out from unreasonable deadlines and commitments.
- E. Consider the system described in question C. Create a workplan that lists the tasks that will need to be completed to meet the project's objectives. Create a Gantt chart and a PERT chart in a project management tool (e.g., Microsoft Project) or using a spreadsheet package to graphically show the high level tasks of the project.
- F. Suppose that you are in charge of the project that is described in question C, and the project will be staffed by members of your class. Do your classmates have all of the right skills to implement such a project? If not,

- how will you go about making sure that the proper skills are available to get the job done?
- G. Consider the application that is used at your school to register for classes. Complete a function point worksheet to determine the size of such an application. You will need to make some assumptions about the application's interfaces and the various factors that affect its complexity.
- H. Read "Your Turn 4-1" near the beginning of this chapter. Create a risk assessment that lists the potential risks associated with performing the project, along with ways to address the risks.
- I. Pretend that your instructor has asked you and two friends to create a Web page to describe the course to potential students and provide current class information (e.g., syllabus, assignments, readings) to current students. You have been assigned the role of leader, so you will need to coordinate your activities and those of your classmates until the project is completed. Describe how you would apply the project management techniques that you have learned in this chapter in this situation. Include descriptions of how you would create a workplan, staff the project, and coordinate all activities—yours and those of your classmates.
- J. Select two project management software packages and research them using the Web or trade magazines. Describe the features of the two packages. If you were a project manager, which one would you use to help support your job? Why?
- K. Select two estimation software packages and research them using the Web or trade magazines. Describe the features of the two packages. If you were a project manager, which one would you use to help support your job? Why?
- L. In 1997, Oxford Health Plans had a computer problem that caused the company to overestimate revenue and underestimate medical costs. Problems were caused by the migration of its claims processing system from the Pick operating system to a UNIX-based system that uses Oracle database software and hardware from Pyramid Technology. As a result, Oxford's stock price plummeted, and fixing the system became the number-one priority for the company. Pretend that you have been placed in charge of managing the repair of the claims processing system. Obviously, the project team will not be in good spirits. How will you motivate team members to meet the project's objectives?

MINICASES

- Emily Pemberton is an IS project manager facing a difficult situation. Emily works for the First Trust Bank, which has recently acquired the City National Bank. Prior to the acquisition, First Trust and City National were bitter rivals, fiercely competing for market share in the region. Following the acrimonious takeover, numerous staff were laid off in many banking areas, including IS. Key individuals were retained from both banks' IS areas, however, and were assigned to a new consolidated IS department. Emily has been made project manager for the first significant IS project since the takeover, and she faces the task of integrating staffers from both banks on her team. The project they are undertaking will be highly visible within the organization, and the time frame for the project is somewhat demanding. Emily believes that the team can meet the project goals successfully, but success will require that the team become cohesive quickly and that potential conflicts are avoided. What strategies do you suggest that Emily implement in order to help ensure a successfully functioning project team?
- Tom, Jan, and Julie are IS majors at Great State University. These students have been assigned a class project

by one of their professors, requiring them to develop a new Web-based system to collect and update information on the IS-program's alumni. This system will be used by the IS graduates to enter job and address information as they graduate, and then make changes to that information as they change jobs and/or addresses. Their professor also has a number of queries that she is interested in being able to implement. Based on their preliminary discussions with their professor, the students have developed this list of system elements:

Inputs: 1 low complexity, 2 medium complexity,
1 high complexity

Outputs: 4 medium complexity

Queries: 1 low complexity, 4 medium complexity,
2 high complexity

Files: 3 medium complexity

Program Interfaces: 2 medium complexity

Assume that an adjusted program complexity of 1.2 is appropriate for this project. Calculate the total adjusted function points for this project.

PART TWO

ANALYSIS PHASE

The Analysis Phase answers the questions of *who* will use the system, *what* the system will do, and *where* and *when* it will be used. During this phase, the project team will learn about the system. The team then produces the Functional Model (Activity Diagrams, Use Case Descriptions and Diagram), Structural Model (CRC Cards and Class and Object Diagrams), and Behavioral Models (Sequence Diagrams, Communication Diagrams, and Behavioral State Machines).

CHAPTER 6 □FUNCTIONAL
MODELING**CHAPTER 7 □**STRUCTURAL
MODELING**CHAPTER 8 □**BEHAVIORAL
MODELING

CHAPTER 5

REQUIREMENTS DETERMINATION

One of the first activities of an analyst is to determine the business requirements for the system. This chapter begins by presenting the requirements definition, a document that lists the new system's capabilities. It then describes how to analyze requirements using business process automation, business process improvement, and business process reengineering techniques, and how to gather requirements using interviews, JAD sessions, questionnaires, document analysis, and observation.

OBJECTIVES

- Understand how to create a requirements definition.
- Become familiar with requirements analysis techniques.
- Understand when to use each requirements analysis technique.
- Understand how to gather requirements using interviews, JAD sessions, questionnaires, document analysis, and observation.
- Understand when to use each requirements-gathering technique.

CHAPTER OUTLINE

Introduction	Interviews
Requirements Determination	Joint Application Development
What is a Requirement?	Questionnaires
Requirements Definition	Document Analysis
Determining Requirements	Observation
Creating the Requirements Definition	Selecting the Appropriate Techniques
Requirements Analysis Techniques	Applying the Concepts at CD Selections
Business Process Automation	Requirements Analysis Techniques
Business Process Improvement	Requirements-Gathering Techniques
Business Process Reengineering	Requirements Definition
Selecting the Appropriate Technique	System Proposal
Requirement-Gathering Techniques	Summary

INTRODUCTION

The systems development life cycle (SDLC) is the process by which the organization moves from the current system (often called the *as-is system*) to the new system (often called the *to-be system*). The output of planning, discussed in Chapters 3 and 4, is the system request, which provides general ideas for the to-be system, defines the project's scope, and provides

the initial workplan. The analysis phase takes the general ideas in the system request and refines them into a detailed requirements definition (this chapter), functional models (Chapter 6), structural models (Chapter 7), and behavioral models (Chapter 8) that together form the system proposal. The *system proposal* also includes revised project management deliverables, such as the feasibility analysis (Chapter 3) and the workplan (Chapter 4).

The system proposal is presented to the approval committee, who decides if the project is to continue. This usually happens at a *system walkthrough*, a meeting at which the concept for the new system is presented to the users, managers, and key decision makers. The goal of the walkthrough is to explain the system in moderate detail so that the users, managers, and key decision makers clearly understand it, can identify needed improvements, and are able to make a decision about whether the project should continue. If approved, the system proposal moves into the design phase, and its elements (requirements definition, functional, structural, and behavioral models) are used as inputs to the steps in design. This further refines them and defines in much more detail how the system will be built.

The line between analysis and design is very blurry. This is because the deliverables created during analysis are really the first step in the design of the new system. Many of the major design decisions for the new system are found in the analysis deliverables. In fact, a better name for analysis would really be “analysis and initial design,” but because this is a rather long name, and because most organizations simply call it analysis, we will too. Nonetheless, it is important to remember that the deliverables from analysis are really the first step in the design of the new system.

In many ways, the requirements determination step is the single most critical step of the entire SDLC, because it is here that the major elements of the system first begin to emerge. During requirements determination, the system is easy to change because little work has been done yet. As the system moves through the other phases in the SDLC, it becomes harder and harder to return to requirements determination and to make major changes because of all of the rework that is involved. Several studies have shown that more than half of all system failures are due to problems with the requirements.¹ This is why the iterative approaches of many object-oriented methodologies are so effective—small batches of requirements can be identified and implemented in incremental stages, allowing the overall system to evolve over time.

In this chapter, we focus on the requirements-determination step of analysis. We begin by explaining what a requirement is and the overall process of requirements gathering and requirements analysis. We then present a set of techniques that can be used to analyze and gather requirements.

REQUIREMENTS DETERMINATION

The purpose of the *requirements determination* step is to turn the very high-level explanation of the business requirements stated in the system request into a more precise list of requirements than can be used as inputs to the rest of analysis (creating functional, structural, and behavioral models). This expansion of the requirements ultimately leads to the design phase.

What is a Requirement?

A *requirement* is simply a statement of what the system must do or what characteristic it must have. During analysis, requirements are written from the perspective of the businessperson, and they focus on the “what” of the system. They focus on business user needs, so they usually are called *business requirements* (and sometimes user requirements). Later

¹ For example, see *The Scope of Software Development Project Failures* by The Standish Group, Dennis MA, 1995.

in design, business requirements evolve to become more technical, and they describe “how” the system will be implemented. Requirements in design are written from the developer’s perspective, and they usually are called *system requirements*.

Before we continue, we want to stress that there is no black-and-white line dividing a business requirement and a system requirement—and some companies use the terms interchangeably. The important thing to remember is that a requirement is a statement of what the system must do, and requirements will change over time as the project moves from analysis to design to implementation. Requirements evolve from detailed statements of the business capabilities that a system should have to detailed statements of the technical way in which the capabilities will be implemented in the new system.

Requirements can be either functional or nonfunctional in nature. A *functional requirement* relates directly to a process the system has to perform or information it needs to contain. For example, requirements that state that the system must have the ability to search for available inventory or to report actual and budgeted expenses are functional requirements. Functional requirements flow directly into the next steps of analysis (functional, structural, and behavioral models) because they define the functions that the system needs to have.

Nonfunctional requirements refer to behavioral properties that the system must have, such as performance and usability. The ability to access the system using a Web browser would be considered a nonfunctional requirement. Nonfunctional requirements may influence the rest of analysis (functional, structural, and behavioral models) but often do so only indirectly; nonfunctional requirements are primarily used in design when decisions are made about the user interface, the hardware and software, and the system’s underlying physical architecture.

Figure 5-1 lists different kinds of nonfunctional requirements and examples of each kind. Notice that the nonfunctional requirements describe a variety of characteristics regarding the system: operational, performance, security, and cultural and political. These characteristics do not describe business processes or information, but they are very important in understanding what the final system should be like. For example, the project team

YOUR TURN

5-1 Identifying Requirements

One of the most common mistakes by new analysts is to confuse functional and nonfunctional requirements. Pretend that you received the following list of requirements for a sales system.

Requirements for Proposed System:
The system should...

1. Be accessible to Web users
2. Include the company standard logo and color scheme
3. Restrict access to profitability information
4. Include actual and budgeted cost information
5. Provide management reports
6. Include sales information that is updated at least daily

7. Have 2-second maximum response time for predefined queries, and 10-minute maximum response time for ad hoc queries
8. Include information from all company subsidiaries
9. Print subsidiary reports in the primary language of the subsidiary
10. Provide monthly rankings of salesperson performance

Questions:

1. Which requirements are functional business requirements? Provide two additional examples.
2. Which requirements are nonfunctional business requirements? What kind of nonfunctional requirements are they? Provide two additional examples.

Nonfunctional Requirement	Description	Examples
Operational	The physical and technical environments in which the system will operate	<ul style="list-style-type: none"> ■ The system should be able to fit in a pocket or purse ■ The system should be able to integrate with the existing inventory system ■ The system should be able to work on any Web browser
Performance	The speed, capacity, and reliability of the system	<ul style="list-style-type: none"> ■ Any interaction between the user and the system should not exceed 2 seconds ■ The system should receive updated inventory information every 15 minutes ■ The system should be available for use 24 hours per day, 365 days per year
Security	Who has authorized access to the system under what circumstances	<ul style="list-style-type: none"> ■ Only direct managers can see personnel records of staff ■ Customers can only see their order history during business hours
Cultural and Political	Cultural, political factors and legal requirements that affect the system	<ul style="list-style-type: none"> ■ The system should be able to distinguish between United States and European currency ■ Company policy says that we only buy computers from Dell ■ Country managers are permitted to authorize customer user interfaces within their units ■ The system shall comply with insurance industry standards

Source: The Atlantic Systems Guild, <http://www.systemsguild.com>

FIGURE 5-1
Nonfunctional Requirements

CONCEPTS

IN ACTION

5-A What Can Happen If You Ignore Nonfunctional Requirements

I once worked on a consulting project in which my manager created a requirements definition without listing non-functional requirements. The project was then estimated based on the requirements definition and sold to the client for \$5,000. In my manager's mind, the system that we would build for the client would be a very simple stand-alone system running on current technology. It shouldn't take more than a week to analyze, design, and build.

Unfortunately, the client had other ideas. They wanted the system to be used by many people in three different departments, and they wanted the ability for any number of people to work on the system concurrently. The technology they had in place was antiquated, but

nonetheless they wanted the system to run effectively on the existing equipment. Because we didn't set the project scope properly by including our assumptions about non-functional requirements in the requirements definition, we basically had to do whatever they wanted.

The capabilities they wanted took weeks to design and program. The project ended up taking four months, and the final project cost was \$250,000. Our company had to pick up the tab for everything except the agreed upon \$5,000. This was by far the most frustrating project situation I ever experienced.

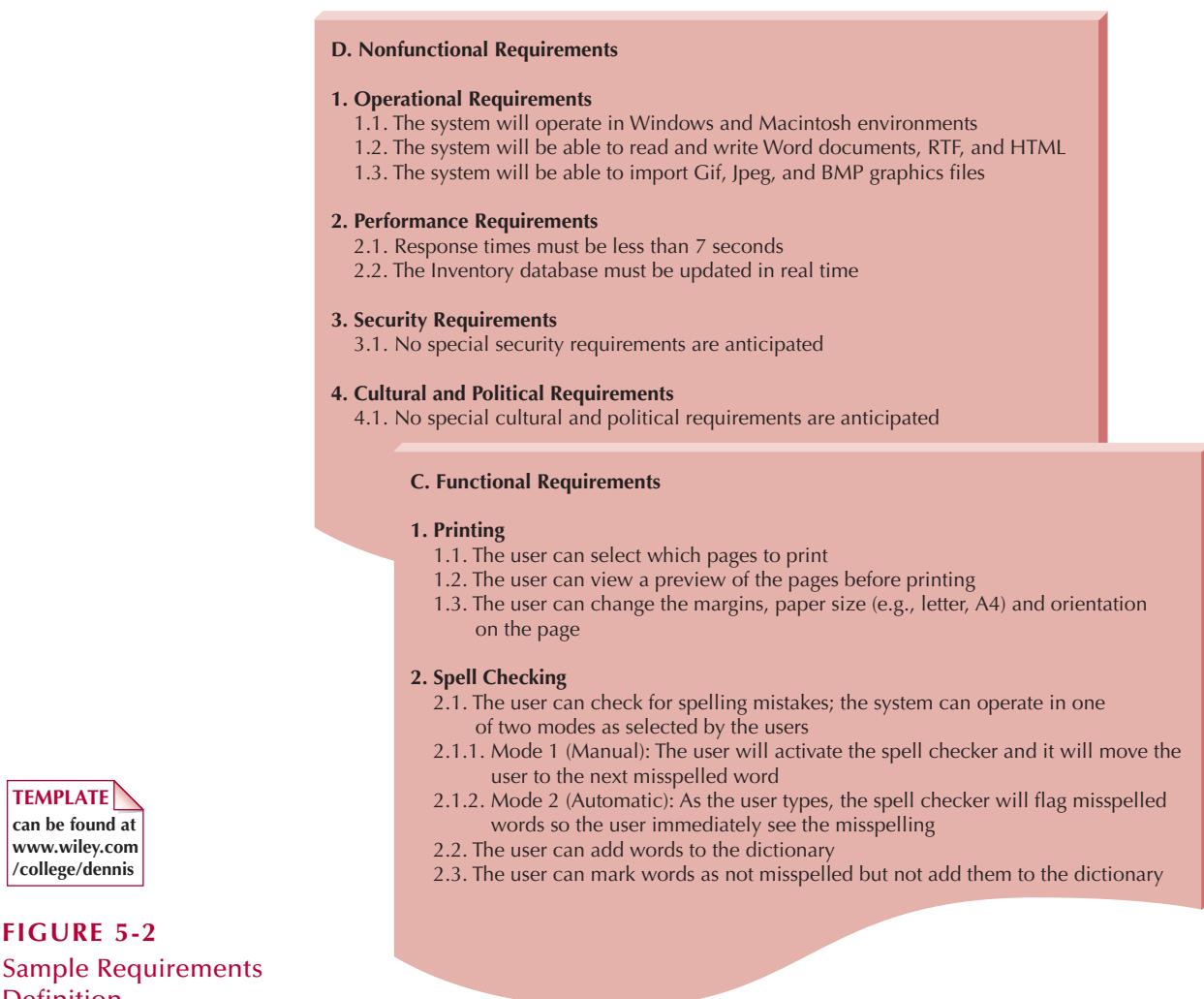
—Barbara Wixom

needs to know if a system needs to be highly secure, requires subsecond response time, or has to reach a multilingual customer base. These requirements will impact design decisions that will be made in design, particularly physical architecture design, so we will revisit them in detail in Chapter 13. The goal at this point is to identify any major issues.

Requirements Definition

The requirements definition report—usually just called the *requirements definition*—is a straightforward text report that simply lists the functional and nonfunctional requirements in an outline format. Figure 5-2 shows a sample requirements definition for a word processing program designed to compete against software, such as Microsoft Word.

The requirements are numbered in a legal or outline format so that each requirement is clearly identified. The requirements are first grouped into functional and nonfunctional requirements and then within each of those headings they are further grouped by the type of nonfunctional requirement or by function.



Sometimes, business requirements are prioritized on the requirements definition. They can be ranked as having high, medium, or low importance in the new system, or they can be labeled with the version of the system that will address the requirement (e.g., release 1, release 2, release 3). This practice is particularly important when using object-oriented methodologies since they deliver requirements in batches by developing incremental versions of the system.

The most *obvious* purpose of the requirements definition is to provide the information needed by the other deliverables in analysis, which include functional, structural, and behavioral models, and to support activities in the design phase. The most *important* purpose of the requirements definition, however, is to define the scope of the system. The document describes to the analysts exactly what the system needs to end up doing. When discrepancies arise, the document serves as the place to go for clarification.

Determining Requirements

Determining requirements for the requirements definition is both a business task and an IT task. In the early days of computing, there was a presumption that the systems analysts, as experts with computer systems, were in the best position to define how a computer system should operate. Many systems failed because they did not adequately address the true business needs of the users. Gradually, the presumption changed so that the users, as the business experts, were seen as being the best position to define how a computer system should operate. However, many systems failed to deliver performance benefits because users simply automated an existing inefficient system, and they failed to incorporate new opportunities offered by technology.

A good analogy is building a house or an apartment. We have all lived in a house or apartment, and most of us have some understanding of what we would like to see in one. However, if we were asked to design one from scratch, it would be a challenge because we lack appropriate design skills and technical engineering skills. Likewise, an architect acting alone would probably miss some of our unique requirements.

Therefore, the most effective approach is to have both businesspeople and analysts working together to determine business requirements. Sometimes, however, users don't know exactly what they want, and analysts need to help them discover their needs. Three kinds of techniques have become popular to help analysts do this: business process automation (BPA), business process improvement (BPI), and business process reengineering (BPR). These techniques are tools that analysts can use when they need to guide the users in explaining what is wanted from a system.

The three kinds of techniques work similarly. They help users critically examine the current state of systems and processes (the *as-is* system), identify exactly what needs to change, and develop a concept for a new system (the *to-be* system). A different amount of change is associated with each technique; BPA creates a small amount of change, BPI creates a moderate amount of change, and BPR creates significant change that affects much of the organization. All three will be described in greater detail later in the chapter.

Although BPA, BPI, and BPR enable the analyst to help users create a vision for the new system, they are not sufficient for extracting information about the detailed business requirements that are needed to build it. Therefore, analysts use a portfolio of requirement-gathering techniques to acquire information from users. The analyst has many gathering techniques from which to choose: interviews, questionnaires, observation, joint application development (JAD), and document analysis. The information gathered using these techniques is critically analyzed and used to craft the requirements definition report. The final section of this chapter describes each of the requirements-gathering techniques in greater depth.

Creating the Requirements Definition

Creating the requirements definition is an iterative and ongoing process whereby the analyst collects information with requirements-gathering techniques (e.g., interviews, document analysis), critically analyzes the information to identify appropriate business requirements for the system, and adds the requirements to the requirements definition report. The requirements definition is kept up to date so that the project team and business users can refer to it and get a clear understanding of the new system.

To create the requirements definition, the project team first determines the kinds of functional and nonfunctional requirements that they will collect about the system (of course, these may change over time). These become the main sections of the document. Next, the analysts use a variety of requirement-gathering techniques (e.g., interviews, observation) to collect information, and they list the business requirements that were identified from that information. Finally, the analysts work with the entire project team and the business users to verify, change, and complete the list and to help prioritize the importance of the requirements that were identified.

This process continues throughout analysis, and the requirements evolves over time as new requirements are identified and as the project moves into later phases of the SDLC. Beware: the evolution of the requirements definition must be carefully managed. The project team cannot keep adding to the requirements definition, or the system will keep growing and growing and never get finished. Instead, the project team carefully identifies requirements and evaluates which ones fit within the scope of the system. When a requirement reflects a real business need but is not within the scope of the current system or current release, it is added on a list of future requirements, or given a low priority. The management of requirements (and system scope) is one of the hardest parts of managing a project!

REQUIREMENTS ANALYSIS TECHNIQUES

Before the project team can determine what requirements are appropriate for a given system, they need to have a clear vision of the kind of system that will be created, and the level of change that it will bring to the organization. The basic process of *analysis* is divided into three steps: understanding the as-is system, identifying improvements, and developing requirements for the to-be system.

Sometimes the first step (i.e., understanding the as-is system) is skipped or done in a cursory manner. This happens when no current system exists, if the existing system and processes are irrelevant to the future system, or if the project team is using a RAD or agile development methodology in which the as-is system is not emphasized. Users of traditional design methods such as waterfall and parallel development (see Chapter 1) typically spend significant time understanding the as-is system and identifying improvements before moving to capture requirements for the to-be system. However, newer RAD, agile, and object-oriented methodologies, such as phased development, prototyping, throwaway prototyping, and extreme programming (see Chapters 1 and 2) focus almost exclusively on improvements and the to-be system requirements, and they spend little time investigating the current as-is system.

Three requirements analysis techniques—business process automation, business process improvement, or business process reengineering—help the analyst lead users through the three (or two) analysis steps so that the vision of the system can be developed. We should note that requirements analysis techniques and requirements-gathering techniques go hand in hand. Analysts need to use requirements-gathering techniques to collect information; requirements analysis techniques drive the kind of information that is gathered and how it is ultimately analyzed. Although we focus now on the analysis techniques

and then discuss requirements gathering at the end of the chapter, they happen concurrently and are complementary activities.

The choice of analysis technique used is based on the amount of change the system is meant to create in the organization. BPA is based on small change that improves process efficiency, BPI creates process improvements that lead to better effectiveness, and BPR revamps the way things work so that the organization is transformed on some level.

To move the users “from here to there,” an analyst needs strong *critical thinking skills*. Critical thinking is the ability to recognize strengths and weaknesses and recast an idea in an improved form, and they are needed to really understand issues and develop new business processes. These skills are needed to thoroughly examine the results of requirements gathering, to identify business requirements, and to translate those requirements into a concept for the new system.

Business Process Automation

Business process automation (BPA) means leaving the basic way in which the organization operates unchanged, and using computer technology to do some of the work. BPA can make the organization more efficient but has the least impact on the business. BPA projects spend a significant time understanding the current as-is system before moving on to improvements and to-be system requirements. Problem analysis and root cause analysis are two popular BPA techniques.

Problem Analysis The most straightforward (and probably the most commonly used) requirements analysis technique is *problem analysis*. Problem analysis means asking the users and managers to identify problems with the as-is system and to describe how to solve them in the to-be system. Most users have a very good idea of the changes they would like to see, and most will be quite vocal about suggesting them. Most changes tend to solve problems rather than capitalize on opportunities, but this is possible, too. Improvements from problem analysis tend to be small and incremental (e.g., provide more space in which to type the customer’s address; provide a new report that currently does not exist).

This type of improvement often is very effective at improving a system’s efficiency or ease of use. However, it often provides only minor improvements in business value—the new system is better than the old, but it may be hard to identify significant monetary benefits from the new system.

Root Cause Analysis The ideas produced by problem analysis tend to be *solutions* to problems. All solutions make assumptions about the nature of the problem, assumptions that may or may not be valid. In our experience, users (and most people in general) tend to jump quickly to solutions without fully considering the nature of the problem. Sometimes the solutions are appropriate, but many times they address a *symptom* of the problem, not the true problem or *root cause* itself.²

For example, suppose you notice that a lightbulb is burned out above your front door. You buy a new bulb, get out a ladder, and replace the bulb. A month later, you see that the same bulb is burnt out, so you buy a new bulb, haul out the ladder, and replace it again. This repeats itself several times. At this point, you have two choices. You can buy a large package of lightbulbs and a fancy lightbulb changer on a long pole so you don’t need the

² Some excellent books that address the importance of gathering requirements and various techniques include: Alan M. Davis, *Software Requirements: Objects, Functions, & States, Revision* (Englewood Cliffs, NJ: Prentice Hall, 1993); Gerald Kotonya and Ian Sommerville, *Requirements Engineering* (Chichester, England: John Wiley & Sons, 1998); and Dean Leffingwell and Don Widrig, *Managing Software Requirements: A Unified Approach* (Reading, MA: Addison-Wesley, 2000).

haul the ladder out each time (thus saving a lot of trips to the store for new bulbs and a lot of effort in working with the ladder). Or you can fix the light fixture that is causing the light to burn out in the first place. Buying the bulb changer is treating the symptom (the burnt-out bulb), while fixing the fixture is treating the root cause.

In the business world, the challenge lies in identifying the root cause—few problems are as simple as the lightbulb problem. The solutions that users propose (or systems that analysts think of) may either address symptoms or root causes, but without a careful analysis, it is difficult to tell which one. And finding out that you've just spent a million dollars on a new lightbulb changer is a horrible feeling!

Root cause analysis therefore focuses on problems, not solutions. The analyst starts by having the users generate a list of problems with the current system, and then prioritize the problems in order of importance. Then starting with the most important, the users and/or the analysts generate all the possible root causes for the problems. Each possible root cause is investigated (starting with the most likely or easiest to check) until the true root cause(s) are identified. If any possible root causes are identified for several problems, those should be investigated first, because there is a good chance they are the real root causes influencing the symptom problems.

In our lightbulb example, there are several possible root causes. A decision tree sometimes helps with the analysis. As Figure 5-3 shows, there are many possible root causes, so buying a new fixture may or may not address the true root cause. In fact, buying a lightbulb changer may actually address the root cause. The key point in root cause analysis is to always challenge the obvious.

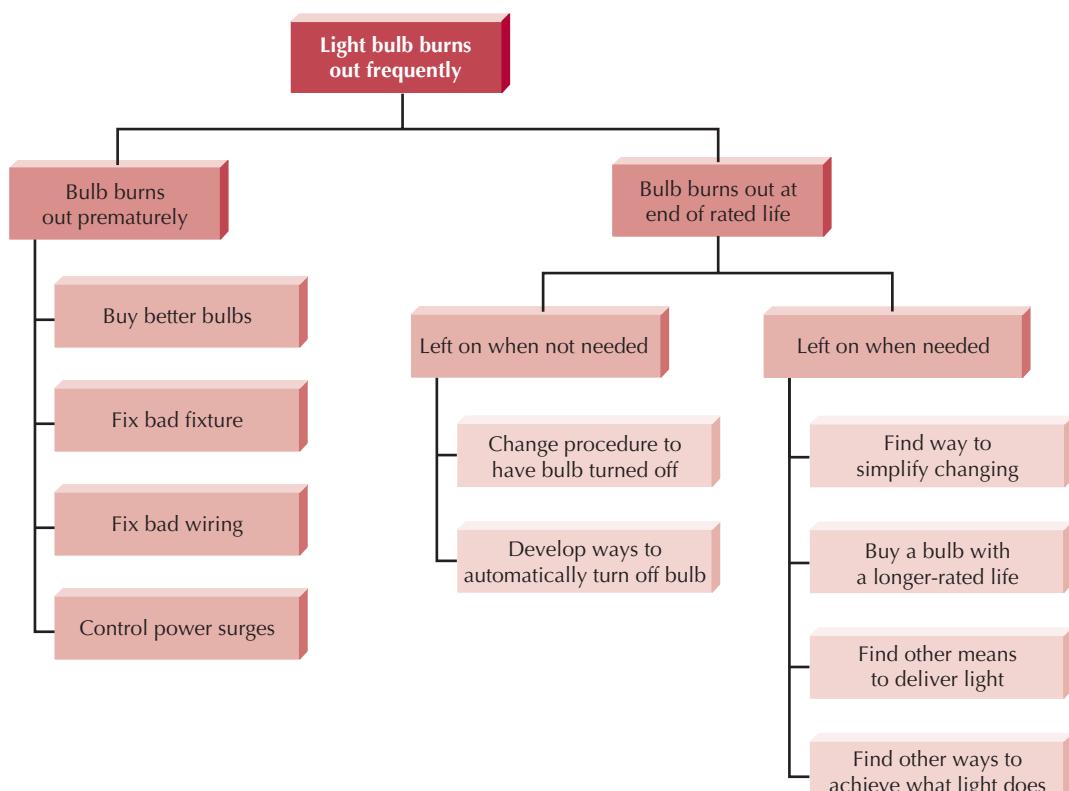


FIGURE 5-3 Root Cause Analysis for the Example of the Burned-Out Lightbulb

Business Process Improvement

Business process improvement (BPI) means making moderate changes to the way in which the organization operates to take advantage of new opportunities offered by technology or to copy what competitors are doing. BPI can improve efficiency (i.e., doing things right) and improve effectiveness (i.e., doing the right things). BPI projects also spend time understanding the as-is system, but much less time than BPA projects; their primary focus is on improving business processes, so time is spent on the as-is only to help with the improvement analyses and the to-be system requirements. Duration analysis, activity-based costing, and information benchmarking are three popular BPI activities.

Duration Analysis *Duration analysis* requires a detailed examination of the amount of time it takes to perform each process in the current as-is system. The analysts begin by determining the total amount of time it takes, on average, to perform a set of business processes for a typical input. They then time each of the individual steps (or subprocesses) in the business process. The time to complete the basic steps are then totaled and compared to the total for the overall process. When there is a significant difference between the two—and in our experiences the total time often can be 10 or even 100 times longer than the sum of the parts—this indicates that this part of the process is badly in need of a major overhaul.

For example, suppose that the analysts are working on a home mortgage system and discover that on average, it takes thirty days for the bank to approve a mortgage. They then look at each of the basic steps in the process (e.g., data entry, credit check, title search, appraisal, etc.) and find that the total amount of time actually spent on each mortgage is

CONCEPTS

IN ACTION

5-B When Optimal Isn't

In 1984, I developed an information system to help schedule production orders in paper mills. Paper is made in huge rolls that are six to ten feet wide. Customer orders (e.g., newspaper, computer paper) are cut from these large rolls. The goal of production scheduling is to decide which orders to combine on one roll to reduce the amount of paper wasted because no matter how you place the orders on the rolls, you almost always end up throwing away the last few inches—customer orders seldom add up to exactly the same width as the roll itself (imagine trying to cut several rolls of toilet paper from a paper towel roll).

The scheduling system was designed to run on an IBM PC, which in those days was not powerful enough to use advanced mathematical techniques like linear programming to calculate which orders to combine on which roll. Instead, we designed the system to use the rules of thumb that the production schedulers themselves had developed over many years. After several months of fine-tuning, the system worked very well. The schedulers were happy and the amount of waste decreased.

By 1986, PCs had increased in power, so we revised the system to use the advanced linear programming tech-

niques to provide better schedules and reduce the waste even more. In our tests, the new system reduced waste by a few percentage points over the original system, so it was installed. However, after several months, there were big problems; the schedulers were unhappy and the amount of waste actually grew.

It turned out that although the new system really did produce better schedules with less waste, the real problem was *not* figuring how to place the orders on the rolls to reduce waste. Instead, the real problem was finding orders for the next week that could be produced with the current batch. The old system combined orders in a way in which the schedulers found it easy to find “matching” future orders. The new system made odd combinations with which the schedulers had a hard time working. The old system was eventually re-installed.

—Alan Dennis

Question

1. How could the problem with the new system have been foreseen?

about eight hours. This is a strong indication that the overall process is badly broken, because it takes thirty days to perform one day's work.

These problems likely occur because the process is badly fragmented. Many different people must perform different activities before the process finishes. In the mortgage example, the application probably sits on many peoples' desks for long periods of time before it is processed. Processes in which many different people work on small parts of the inputs are prime candidates for *process integration* or *parallelization*. Process integration means changing the fundamental process so that fewer people work on the input, which often requires changing the processes and retraining staff to perform a wider range of duties. Process parallelization means changing the process so that all the individual steps are performed at the same time. For example in the mortgage application example, there is probably no reason that the credit check cannot be performed at the same time as the appraisal and title check.

Activity-Based Costing *Activity-based costing* is a similar analysis that examines the cost of each major process or step in a business process rather than the time taken.³ The analysts identify the costs associated with each of the basic functional steps or processes, identify the most costly processes, and focus their improvement efforts on them.

Assigning costs is conceptually simple. You just examine the direct cost of labor and materials for each input. Materials costs are easily assigned in a manufacturing process, while labor costs are usually calculated based on the amount of time spent on the input and the hourly cost of the staff. However, as you may recall from a managerial accounting course, there are indirect costs such as rent, depreciation, and so on that also can be included in activity costs.

Informal Benchmarking *Benchmarking* refers to studying how other organizations perform a business process in order to learn how your organization can do something better. Benchmarking helps the organization by introducing ideas that employees may never have considered, but have the potential to add value.

Informal benchmarking is fairly common for "customer-facing" business processes (i.e., those processes that interact with the customer). With informal benchmarking, the

CONCEPTS

5-C Duration Analysis

IN ACTION

A group of executives from a *Fortune* 500 company used Duration Analysis to discuss their procurement process. Using a huge wall of Velcro and a handful of placards, a facilitator proceeded to map out the company's process for procuring a \$50 software upgrade. Having quantified the time it took to complete each step, she then assigned costs based on the salaries of the employees involved. The

15-minute exercise left the group stunned. Their procurement process had gotten so convoluted that it took eighteen days, countless hours of paperwork and nearly \$22,000 in people time to get the product ordered, received, and up and running on the requester's desktop.

Source: "For Good Measure," *CIO Magazine*, March 1 1999, by Debby Young.

³ For more information on activity-based costing, see K.B. Burk and D. W. Webster, *Activity Based Costing* (Fairfax, VA: American Management Systems, 1994); and D. T. Hicks, *Activity-Based Costing: Making It Work for Small and Mid-Sized Companies* (New York: John Wiley, 1998).

managers and analysts think about other organizations, or visit them as customers to watch how the business process is performed. In many cases, the business studied may be a known leader in the industry or simply a related firm. For example, suppose the team is developing a Web site for a car dealer. The project sponsor, key managers, and key team members would likely visit the Web sites of competitors, as well as those of others in the car industry (e.g., manufacturers, accessories suppliers) and those in other industries that have won awards for their Web sites.

Business Process Reengineering

Business process reengineering (BPR) means changing the fundamental way in which the organization operates—“obliterating” the current way of doing business and making major changes to take advantage of new ideas and new technology. BPR projects spend little time understanding the *as-is*, because their goal is to focus on new ideas and new ways of doing business. Outcome analysis, technology analysis, and activity elimination are three popular BPR activities.

Outcome Analysis *Outcome analysis* focuses on understanding the fundamental outcomes that provide value to customers. While these outcomes sound as though they should be obvious, they often aren’t. For example, suppose you are an insurance company, and one of your customers has just had a car accident. What is the fundamental outcome from the *customer’s* perspective? Traditionally, insurance companies have answered this question by assuming the customer wants to receive the insurance payment quickly. To the customer, however, the payment is only a *means* to the real outcome: a repaired car. The insurance company might benefit by extending their view of the business process past its traditional boundaries to include not paying for repairs, but performing the repairs or contracting with an authorized body shop to do them.

With this approach, the system analysts encourage the managers and project sponsor to pretend they are customers and to think carefully about what the organization’s products and services enable the customers to do—and what they *could* enable the customer to do.

Technology Analysis Many major changes in business over the past decade have been enabled by new technologies. *Technology analysis* therefore starts by having the analysts and managers develop a list of important and interesting technologies. Then the group systematically identifies how each and every technology could be applied to the business process and identifies how the business would benefit.

For example, one useful technology might be the Internet. Saturn, the car manufacturer, took this idea and developed an Extranet application for its suppliers. Rather than ordering parts for its cars, Saturn makes its production schedule available electronically to its suppliers, who ship the parts Saturn needs so that they arrive at the plant just in time. This saves Saturn significant costs because it eliminates the need for people to monitor the production schedule and issue purchase orders.

Activity Elimination *Activity elimination* is exactly what it sounds like. The analysts and managers work together to identify how the organization could eliminate each and every activity in the business process, how the function could operate without it, and what effects are likely to occur. Initially, managers are reluctant to conclude that processes can be eliminated, but this is a “force-fit” exercise in that they must eliminate each activity. In some cases the results are silly, but nonetheless, participants must address each and every activity in the business process.

For example, in the home mortgage approval process discussed earlier, the managers and analysts would start by eliminating the first activity, entering the data into the mortgage company's computer. This leads to two obvious possibilities (1) eliminate the use of a computer system; or (2) make someone else do the data entry (e.g., the customer over the Web). They would then eliminate the next activity, the credit check. Silly right? After all, making sure the applicant has good credit is critical in issuing a loan. Not really. The real answer depends upon how many times the credit check identifies bad applications. If all or almost all applicants have good credit and are seldom turned down by a credit check, then the cost of the credit check may not be worth the cost of the few bad loans it prevents. Eliminating it may actually result in lower costs even with the cost of bad loans.

Selecting the Appropriate Technique

Each of the techniques discussed in this chapter has its own strengths and weaknesses (see Figure 5-4). No one technique is inherently better than the others, and in practice most projects use a combination of techniques.

Potential Business Value The *potential business value* varies with analysis strategy. While BPA has the potential to improve the business, most of the benefits from BPA are tactical and small in nature. Since BPA does not seek to change the business processes, it can only improve their efficiency. BPI usually offers moderate potential benefits, depending upon the scope of the project, because it seeks to change the business in some way. It can increase both efficiency and effectiveness. BPR creates large *potential* benefits because it seeks to radically improve the nature of the business.

Project Cost *Project cost* is always important. In general, BPA requires the lowest cost because it has the narrowest focus and seeks to make the fewest number of changes. BPI can be moderately expensive, depending upon the scope of the project. BPR is usually expensive, both because of the amount of time required of senior managers and the amount of redesign to business processes.

Breadth of Analysis *Breadth of analysis* refers to the scope of analysis, or whether analysis includes business processes within a single business function, processes that cross the organization, or processes that interact with those in customer or supplier organizations. BPR takes a broad perspective, often spanning several major business processes, even across multiple organizations. BPI has a much narrower scope that usually includes one or several business functions. BPA typically examines a single process.

Risk One final issue is *risk* of failure, which is the likelihood of failure due to poor design, unmet needs, or too much change for the organization to handle. BPA and BPI have low to moderate risk because the to-be system is fairly well defined and understood,

	Business Process Automation	Business Process Improvement	Business Process Reengineering
Potential business value	Low–moderate	Moderate	High
Project cost	Low	Low–moderate	High
Breadth of analysis	Narrow	Narrow–moderate	Very broad
Risk	Low–moderate	Low–moderate	Very high

FIGURE 5-4
Characteristics of Analysis Strategies

and its potential impact on the business can be assessed before it is implemented. BPR projects, on the other hand, are less predictable. BPR is extremely risky and not something to be undertaken unless the organization and its senior leadership are committed to making significant changes. Mike Hammer, the father of BPR, estimates that 70 percent of BPR projects fail.

YOUR TURN

5-2 IBM Credit

IBM Credit was a wholly owned subsidiary of IBM responsible for financing mainframe computers sold by IBM. While some customers bought mainframes outright, or obtained financing from other sources, financing computers provided significant additional profit.

When an IBM sales representative made a sale, he or she would immediately call IBM Credit to obtain a financing quote. The call was received by a credit officer who would record the information on a request form. The form would then be sent to the credit department to check the customer's credit status. This information would be recorded on the form, which was then sent to the business practices department who would write a contract (sometimes reflecting changes requested by the customer). The form and the contract would then go to the pricing department, which used the credit information to establish an interest rate and recorded it on the Form. The Form and contract was then sent to the clerical group, where an administrator would prepare a cover letter quoting the interest rate and send the letter and contract via Federal Express to the customer.

The problem at IBM Credit was a major one. Getting a financing quote took anywhere for four to eight days (six days on average), giving the customer time to rethink the order or find financing elsewhere. While the quote was being prepared, sales representatives would often call to find out where the quote was in the process, so they could tell the customer when to expect it. However, no one at IBM Credit could answer the question because the paper forms could be in any department and it was impossible to locate one without physically walking

through the departments and going through the piles of forms on everyone's desk.

IBM Credit examined the process and changed it so that each credit request was logged into a computer system so that each department could record an application's status as they completed it and sent it to the next department. In this way, sales representatives could call the credit office and quickly learn the status of each application. IBM used some sophisticated management science queuing theory analysis to balance workloads and staff across the different departments so none would be overloaded. They also introduced performance standards for each department (e.g., the pricing decision had to be completed within one day after that department received an application).

However, process times got worse, even though each department was achieving almost 100 percent compliance on its performance goals. After some investigation, managers found that when people got busy, they conveniently found errors that forced them to return credit requests to the previous department for correction, thereby removing it from their time measurements.

Questions:

1. What techniques can you use to identify improvements? Choose one technique and apply it to this situation—what improvements did you identify?

Source: *Reengineering the Corporation*, New York: Harper Business, 1993, by M. Hammer and J. Champy.

YOUR TURN

5-3 Analysis Strategy

Suppose you are the analyst charged with developing a new Web site for a local car dealer who wants to be very

innovative and try new things. What analysis techniques would you recommend? Why?

REQUIREMENTS-GATHERING TECHNIQUES

An analyst is very much like a detective (and business users sometimes are like elusive suspects). He or she knows that there is a problem to be solved and therefore must look for clues that uncover the solution. Unfortunately, the clues are not always obvious (and often missed), so the analyst needs to notice details, talk with witnesses, and follow leads just as Sherlock Holmes would have done. The best analysts will thoroughly gather requirements using a variety of techniques and make sure that the current business processes and the needs for the new system are well understood before moving into design. You don't want to discover later that you have key requirements wrong—surprises like this late in the SDLC can cause all kinds of problems.

The requirements-gathering process is used for building political support for the project and establishing trust and rapport between the project team building the system and the users who ultimately will choose to use or not use the system. Involving someone in the process implies that the project teams views that person as an important resource and values his or her opinions. You *must* include all of the key *stakeholders* (the people who can affect the system or who will be affected by the system) in the requirements-gathering process. This might include managers, employees, staff members, and even some customers and suppliers. If you do not involve a key person, that individual may feel slighted, which can cause problems during implementation (e.g., "How could they have developed the system without my input?!").

The second challenge of requirements gathering is choosing the way(s) in which information is collected. There are many techniques for gathering requirements that vary from asking people questions to watching them work. In this chapter, we focus on the five most commonly used techniques: interviews, JAD sessions (a special type of group meeting), questionnaires, document analysis, and observation. Each technique has its own strengths and weaknesses, many of which are complementary, so most projects use a combination of techniques, probably most often interviews, JAD sessions, and document analysis.⁴

Interviews

The *interview* is the most commonly used requirements-gathering technique. After all, it is natural—usually if you need to know something, you ask someone. In general, interviews are conducted one-on-one (one interviewer and one interviewee), but sometimes, due to time constraints, several people are interviewed at the same time. There are five basic steps to the interview process: selecting interviewees, designing interview questions, preparing for the interview, conducting the interview, and postinterview follow-up.⁵

Selecting Interviewees The first step to interviewing is to create an *interview schedule* that lists all of the people who will be interviewed, when, and for what purpose (see Figure 5-5). The schedule can be an informal list that is used to help set up meeting times, or a formal list that is incorporated into the workplan. The people who appear on the interview schedule are selected based on the analyst's information needs. The project sponsor, key business users, and other members of the project team can help the analyst determine who in the organization can best provide important information about requirements. These people are listed on the interview schedule in the order in which they should be interviewed.

⁴ Two good books that discuss the problems in finding the root causes to problems are: E.M. Goldratt, and J. Cox, *The Goal* (Croton-on-Hudson, NY: North River Press, 1986), and E.M. Goldratt, *The Haystack Syndrome* (Croton-on-Hudson, NY: North River Press, 1990).

⁵ Two good books on interviewing are Brian James, *The Systems Analysis Interview* (Manchester: NCC Blackwell, 1989); and James P. Spradley, *The Ethnographic Interview* (New York, NY: Holt, Rinehart, and Winston, 1979).

FIGURE 5-5
Sample Interview Schedule

Name	Position	Purpose of Interview	Meeting
Andria McClellan	Director, Accounting	Strategic vision for new accounting system	Mon, March 1 8:00–10:00 AM
Jennifer Draper	Manager, Accounts Receivable	Current problems with accounts receivable process; future goals	Mon, March 1 2:00–3:15 PM
Mark Goodin	Manager, Accounts Payable	Current problems with accounts payable process; future goals	Mon, March 1 4:00–5:15 PM
Anne Asher	Supervisor, Data Entry	Accounts receivable and payable processes	Wed, March 3 10:00–11:00 AM
Fernando Merce	Data Entry Clerk	Accounts receivable and payable processes	Wed, March 3 1:00–3:00 PM

**CONCEPTS
IN ACTION**

5-D Selecting the Wrong People

In 1990, I led a consulting team for a major development project for the U.S. Army. The goal was to replace eight existing systems used on virtually every Army base across the United States. The as-is analysis models for these systems had been built, and our job was to identify improvement opportunities and develop to-be process models for each of the eight systems.

For the first system, we selected a group of mid-level managers (captains and majors) recommended by their commanders as being the experts in the system under construction. These individuals were the first and second line

managers of the business function. The individuals were expert at managing the process, but did not know the exact details of how the process worked. The resulting to-be analysis models were very general and non-specific.

—Alan Dennis

Question

1. Suppose you were in charge of the project. Create an interview schedule for the remaining seven projects.

People at different levels of the organization will have different perspectives on the system, so it is important to include both managers who manage the processes and staff who actually perform the processes to gain both high-level and low-level perspectives on an issue. Also, the kinds of interview subjects that you need may change over time. For example, at the start of the project, the analyst has a limited understanding of the as-is business process. It is common to begin by interviewing one or two senior managers to get a strategic view, and then move to mid-level managers who can provide broad, overarching information about the business process and the expected role of the system being developed. Once the analyst has a good understanding of the “big picture,” lower-level managers and staff members can fill in the exact details of how the process works. Like most other things about systems analysis, this is an iterative process—starting with senior managers, moving to mid-level managers, then staff members, back to mid-level managers, and so on, depending upon what information is needed along the way.

It is quite common for the list of interviewees to grow, often by 50 percent to 75 percent. As you interview people, you likely will identify more information that is needed and additional people who can provide the information.

Designing Interview Questions There are three types of interview questions: closed-ended questions, open-ended questions, and probing questions. *Closed-ended questions* are those that require a specific answer. You can think of them as being similar to multiple choice or arithmetic questions on an exam (see Figure 5-6). Closed-ended questions are used when the analyst is looking for specific, precise information (e.g., how many credit card requests are received per day). In general, precise questions are best. For example, rather than asking “Do you handle a lot of requests?” it is better to ask “How many requests do you process per day?”

Closed-ended questions enable analysts to control the interview and obtain the information they need. However, these types of questions don’t uncover *why* the answer is the way it is, nor do they uncover information that the interviewer does not think to ask ahead of time.

Open-ended questions are those that leave room for elaboration on the part of the interviewee. They are similar in many ways to essay questions that you might find on an exam (see Figure 5-6 for examples). Open-ended questions are designed to gather rich information and give the interviewee more control over the information that is revealed during the interview. Sometimes the information that the interviewee chooses to discuss uncovers information that is just as important as the answer (e.g., if the interviewee talks only about other departments when asked for problems, it may suggest that he or she is reluctant to admit his or her own problems).

The third type of question is the *probing question*. Probing questions follow-up on what has just been discussed in order to learn more, and they often are used when the interviewer is unclear about an interviewee’s answer. They encourage the interviewee to expand on or to confirm information from a previous response, and they are a signal that the interviewer is listening and interested in the topic under discussion. Many beginning analysts are reluctant to use probing questions because they are afraid that the interviewee might be offended at being challenged or because they believe it shows that they didn’t understand what the interviewee said. When done politely, probing questions can be a powerful tool in requirements gathering.

In general, you should not ask questions about information that is readily available from other sources. For example, rather than asking what information is used to perform a task, it is simpler to show the interviewee a form or report (see document analysis later) and ask what information on it is used. This helps focus the interviewee on the task, and saves time, because he or she does not need to describe the information detail—he or she just needs to point it out on the form or report.

Types of Questions	Examples
Closed-Ended Questions	<ul style="list-style-type: none"> • How many telephone orders are received per day? • How do customers place orders? • What information is missing from the monthly sales report?
Open-Ended Questions	<ul style="list-style-type: none"> • What do you think about the current system? • What are some of the problems you face on a daily basis? • What are some of the improvements you would like to see in a new system?
Probing Questions	<ul style="list-style-type: none"> • Why? • Can you give me an example? • Can you explain that in a bit more detail?

FIGURE 5-6
Three Types of
Questions

No question type is better than another, and usually a combination of questions is used during an interview. At the initial stage of an IS development project, the as-is process can be unclear, so the interview process begins with *unstructured interviews*, interviews that seek a broad and roughly defined set of information. In this case, the interviewer has a general sense of the information needed, but few close-ended questions to ask. These are the most challenging interviews to conduct because they require the interviewer to ask open-ended questions and probe for important information “on the fly.”

As the project progresses, the analyst comes to understand the business process much better, and he or she needs very specific information about how business processes are performed (e.g., exactly how a customer credit card is approved). At this time, the analyst conducts *structured interviews*, in which specific sets of questions are developed prior to the interviews. There usually are more close-ended questions in a structured interview than in the unstructured approach.

No matter what kind of interview is being conducted, interview questions must be organized into a logical sequence, so that the interview flows well. For example, when trying to gather information about the current business process, it can be useful to move in logical order through the process or from the most important issues to the least important.

There are two fundamental approaches to organizing the interview questions: top-down or bottom-up; see Figure 5-7. With the *top-down interview*, the interviewer starts with broad, general issues and gradually works towards more specific ones. With the *bottom-up interview*, the interviewer starts with very specific questions and moves to broad questions. In practice, analysts mix the two approaches, starting with broad general issues, moving to specific questions, and then back to general issues.

The top-down approach is an appropriate strategy for most interviews (it is certainly the most common approach). The top-down approach enables the interviewee to become accustomed to the topic before he or she needs to provide specifics. It also enables the interviewer to understand the issues before moving to the details because the interviewer may not have sufficient information at the start of the interview to ask very specific questions. Perhaps most importantly, the top-down approach enables the interviewee to raise a set of “big picture” issues before becoming enmeshed in details, so the interviewer is less likely to miss important issues.

One case in which the bottom-up strategy may be preferred is when the analyst already has gathered a lot of information about issues and just needs to fill in some holes with

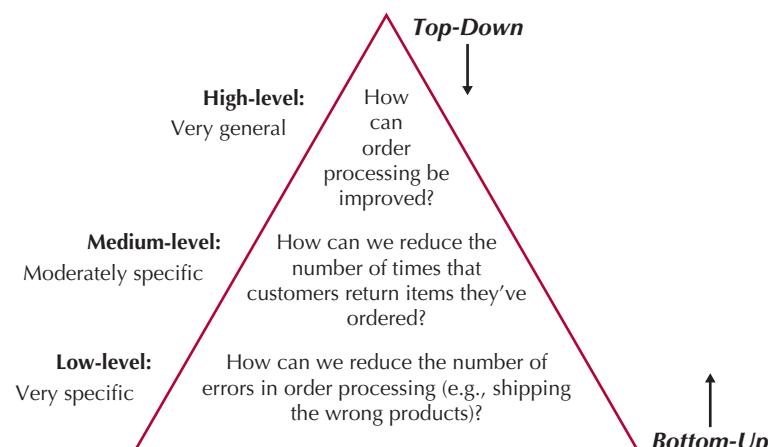


FIGURE 5-7
Top-Down and Bottom-Up Questioning Strategies

details. Or, bottom-up may be appropriate if lower-level staff members are threatened or unable to answer high-level questions. For example, “How can we improve customer service?” may be too broad a question for a customer service clerk, whereas a specific question is readily answerable (e.g., “How can we speed up customer returns?”). In any event, all interviews should begin with noncontroversial questions first, and then gradually move into more contentious issues after the interviewer has developed some rapport with the interviewee.

Preparing for the Interview It is important to prepare for the interview in the same way that you would prepare to give a presentation. You should have a general interview plan that lists the questions that you will ask in the appropriate order; anticipates possible answers and provides how you will follow up with them; and identifies segues between related topics. Confirm the areas in which the interviewee has knowledge so you do not ask questions that he or she cannot answer. Review the topic areas, the questions, and the interview plan, and clearly decide which have the greatest priority in case you run out of time.

In general, structured interviews with closed-ended questions take more time to prepare than unstructured interviews. So, some beginning analysts prefer unstructured interviews, thinking that they can “wing it.” This is very dangerous and often counterproductive, because any information not gathered in the first interview would require follow-up efforts, and most users do not like to be interviewed repeatedly about the same issues.

PRACTICAL

TIP

5-1 Developing Interpersonal Skills



Interpersonal skills are those skills than enable you to develop rapport with others, and they are very important for interviewing. They help you to communicate with others effectively. Some people develop good interpersonal skills at an early age; they simply seem to know how to communicate and interact with others. Other people are less “lucky” and need to work hard to develop their skills. Interpersonal skills, like most skills, can be learned. Here are some tips:

- **Don’t worry, be happy.** Happy people radiate confidence and project their feelings on others. Try interviewing someone while smiling and then interviewing someone else while frowning and see what happens!
- **Pay attention.** Pay attention to what the other person is saying (which is harder than you might think). See how many times you catch yourself with your mind on something other than the conversation at hand.
- **Summarize key points.** At the end of each major theme or idea that someone explains, you should

repeat the key points back to the speaker (e.g., “Let me make sure I understand. The key issues are...”). This demonstrates that you consider the information important—and also forces you to pay attention (you can’t repeat what you didn’t hear).

- **Be succinct.** When you speak, be succinct. The goal in interviewing (and in much of life) is to learn, not to impress. The more you speak, the less time you give to others.
- **Be honest.** Answer all questions truthfully, and if you don’t know the answer, say so.
- **Watch body language (yours and theirs).** The way a person sits or stands conveys much information. In general, a person who is interested in what you are saying sits or leans forward, makes eye contact, and often touches his or her face. A person leaning away from you or with an arm over the back of a chair is disinterested. Crossed arms indicate defensiveness or uncertainty, while “steepling” (sitting with hands raised in front of the body with fingertips touching) indicates a feeling of superiority.

Be sure to prepare the interviewee as well. When you schedule the interview, inform the interviewee of the reason for the interview and the areas you will be discussing far enough in advance so that he or she has time to think about the issues and organize his or her thoughts. This is particularly important when you are an outsider to the organization, and for lower-level employees who often are not asked for their opinions and who may be uncertain about why you are interviewing them.

Conducting the Interview When you start the interview, the first goal is to build rapport with the interviewee, so that he or she trusts you and is willing to tell you the whole truth, not just give the answers that he or she thinks you want. You should appear to be professional and an unbiased, independent seeker of information. The interview should start with an explanation of why you are there and why you have chosen to interview the person, and then move into your planned interview questions.

It is critical to carefully record all the information that the interviewee provides. In our experience, the best approach is to take careful notes—write down *everything* the interviewee says, even if it does not appear immediately relevant. Don't be afraid to ask the person to slow down or to pause while you write, because this is a clear indication that the interviewee's information is important to you. One potentially controversial issue is whether or not to tape-record the interview. Recording ensures that you do not miss important points, but it can be intimidating for the interviewee. Most organizations have policies or generally accepted practices about the recording of interviews, so find out what they are before you start an interview. If you are worried about missing information and cannot tape the interview, then bring along a second person to take detailed notes.

As the interview progresses, it is important that you understand the issues that are discussed. If you do not understand something, be sure to ask. Don't be afraid to ask "dumb questions" because the only thing worse than appearing "dumb" is to be "dumb" by not understanding something. If you don't understand something during the interview, you certainly won't understand it afterward. Try to recognize and define jargon, and be sure to clarify jargon you do not understand. One good strategy to increase your understanding during an interview is to periodically summarize the key points that the interviewee is communicating. This avoids misunderstandings and also demonstrates that you are listening.

Finally, be sure to separate facts from opinion. The interviewee may say, for example, "we process too many credit card requests." This is an opinion, and it is useful to follow this up with a probing question requesting support for the statement (e.g., "Oh, how many do you process in a day?"). It is helpful to check the facts because any differences between the facts and the interviewee's opinions can point out key areas for improvement. Suppose the interviewee complains about a high or increasing number of errors, but the logs show that errors have been decreasing. This suggests that errors are viewed as a very important problem that should be addressed by the new system, even if they are declining.

As the interview draws to a close, be sure to give the interviewee time to ask questions or provide information that he or she thinks is important but was not part of your interview plan. In most cases, the interviewee will have no additional concerns or information, but in some cases this will lead to unanticipated, but important information. Likewise, it can be useful to ask the interviewee if there are other people who should be interviewed. Make sure that the interview ends on time (if necessary, omit some topics or plan to schedule another interview).

As a last step in the interview, briefly explain what will happen next (see the next section). You don't want to prematurely promise certain features in the new system or a specific delivery date, but you do want to reassure the interviewee that his or her time was well spent and very helpful to the project.

Post-Interview Follow-up After the interview is over, the analyst needs to prepare an *interview report* that describes the information from the interview (Figure 5-8). The report contains *interview notes*, information that was collected over the course of the interview and is summarized in a useful format. In general, the interview report should be written within forty-eight hours of the interview, because the longer you wait, the more likely you are to forget information.

Often, the interview report is sent to the interviewee with a request to read it and inform the analyst of clarifications or updates. Make sure the interviewee is convinced that you genuinely want his or her corrections to the report. Usually there are few changes, but the need for any significant changes suggests that a second interview will be required. Never distribute someone's information without prior approval.

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 5-8
Interview Report

Interview Notes Approved By: Linda Estey
<p>Person Interviewed: Linda Estey, Director, Human Resources</p> <p>Interviewer: Barbara Wixom</p> <p>Purpose of Interview:</p> <ul style="list-style-type: none"> • Understand reports produced for Human Resources by the current system • Determine information requirements for future system <p>Summary of Interview:</p> <ul style="list-style-type: none"> • Sample reports of all current HR reports are attached to this report. The information that is not used and missing information are noted on the reports. • Two biggest problems with the current system are: <ol style="list-style-type: none"> 1. The data is too old (the HR Department needs information within two days of month end; currently information is provided to them after a three-week delay) 2. The data is of poor quality (often reports must be reconciled with departmental HR database) • The most common data errors found in the current system include incorrect job level information and missing salary information. <p>Open Items:</p> <ul style="list-style-type: none"> • Get current employee roster report from Mary Skudrna (extension 4355). • Verify calculations used to determine vacation time with Mary Skudrna. • Schedule interview with Jim Wack (extension 2337) regarding the reasons for data quality problems. <p>Detailed Notes: See attached transcript.</p>

YOUR TURN

5-4 Interview Practice

Interviewing is not as simple as it first appears. Select two people from class to go to the front of the room to demonstrate an interview. (This also can be done in groups.) Have one person be the interviewer, and the other the interviewee. The interviewer should conduct a 5-minute interview regarding the school course registration system. Gather information about the existing system and how the system can be improved. If there is time, repeat with another pair.

Questions:

1. Describe the body language of the interview pair.
2. What kind of interview was conducted?
3. What kinds of questions were asked?
4. What was done well? How could the interview be improved?

Joint Application Development(JAD)

Joint application development (or *JAD* as it is more commonly known) is an information gathering technique that allows the project team, users, and management to work together to identify requirements for the system. IBM developed the *JAD* technique in the late 1970s, and it is often the most useful method for collecting information from users.⁶ Capers Jones claims that *JAD* can reduce scope creep by 50 percent, and it avoids the requirements for a system from being too specific or too vague, both of which cause trouble during later stages of the *SDLC*.⁷ *JAD* is a structured process in which ten to twenty users meet together under the direction of *facilitator* skilled in *JAD* techniques. The facilitator is a person who sets the meeting agenda and guides the discussion, but does not join in the discussion as a participant. He or she does not provide ideas or opinions on the topics under discussion to remain neutral during the session. The facilitator must be an expert in both group process techniques and systems analysis and design techniques. One or two *scribes* assist the facilitator by recording notes, making copies, and so on. Often the scribes will use computers and *CASE* tools to record information as the *JAD* session proceeds.

The *JAD* group meets for several hours, several days, or several weeks until all of the issues have been discussed and the needed information is collected. Most *JAD* sessions take place in a specially prepared meeting room, away from the participants' offices so that they are not interrupted. The meeting room is usually arranged in a U shape so that all participants can easily see each other (see Figure 5-9). At the front of the room (the open part of the "U"), there is a whiteboard, flip chart and/or overhead projector for use by the facilitator who leads the discussion.

One problem with *JAD* is that it suffers from the traditional problems associated with groups; sometimes people are reluctant to challenge the opinions of others (particularly their boss), a few people often dominate the discussion, and not everyone participates. In a fifteen-member group, for example, if everyone participates equally, then each person can talk for only four minutes each hour and must listen for the remaining fifty-six minutes—not a very efficient way to collect information.

A new form of *JAD* called *electronic JAD* or *e-JAD* attempts to overcome these problems by using groupware. In an *e-JAD* meeting room, each participant uses special software on a networked computer to send anonymous ideas and opinions to everyone else. In this way, all participants can contribute at the same time, without fear of reprisal from people with differing opinions. Initial research suggests that *e-JAD* can reduce the time required to run *JAD* sessions by 50 percent to 80 percent.⁸

Selecting Participants Selecting *JAD* participants is done in the same basic way as selecting interview participants. Participants are selected based on the information they can contribute, to provide a broad mix of organizational levels, and to build political support for the new system. The need for all *JAD* participants to be away from their office at the same time can be a major problem. The office may need to be closed or run with a "skeleton" staff until the *JAD* sessions are complete.

⁶ More information on *JAD* can be found in J. Wood and D. Silver, *Joint Application Development* (New York: John Wiley & Sons, 1989); and Alan Cline, "Joint Application Development for Requirements Collection and Management," <http://www.carolla.com/wp-jad.htm>.

⁷ See Kevin Strehlo, "Catching up with the Jones and 'Requirement' Creep," *InfoWorld*, July 29, 1996, and Kevin Strehlo, "The Makings of a Happy Customer: Specifying Project X" *InfoWorld*. Nov 11, 1996.

⁸ For more information on *e-JAD*, see A. R. Dennis, G. S. Hayes, and R. M. Daniels, "Business Process Modeling with Groupware," *Journal of MIS* 15(4), 1999, 115–142.

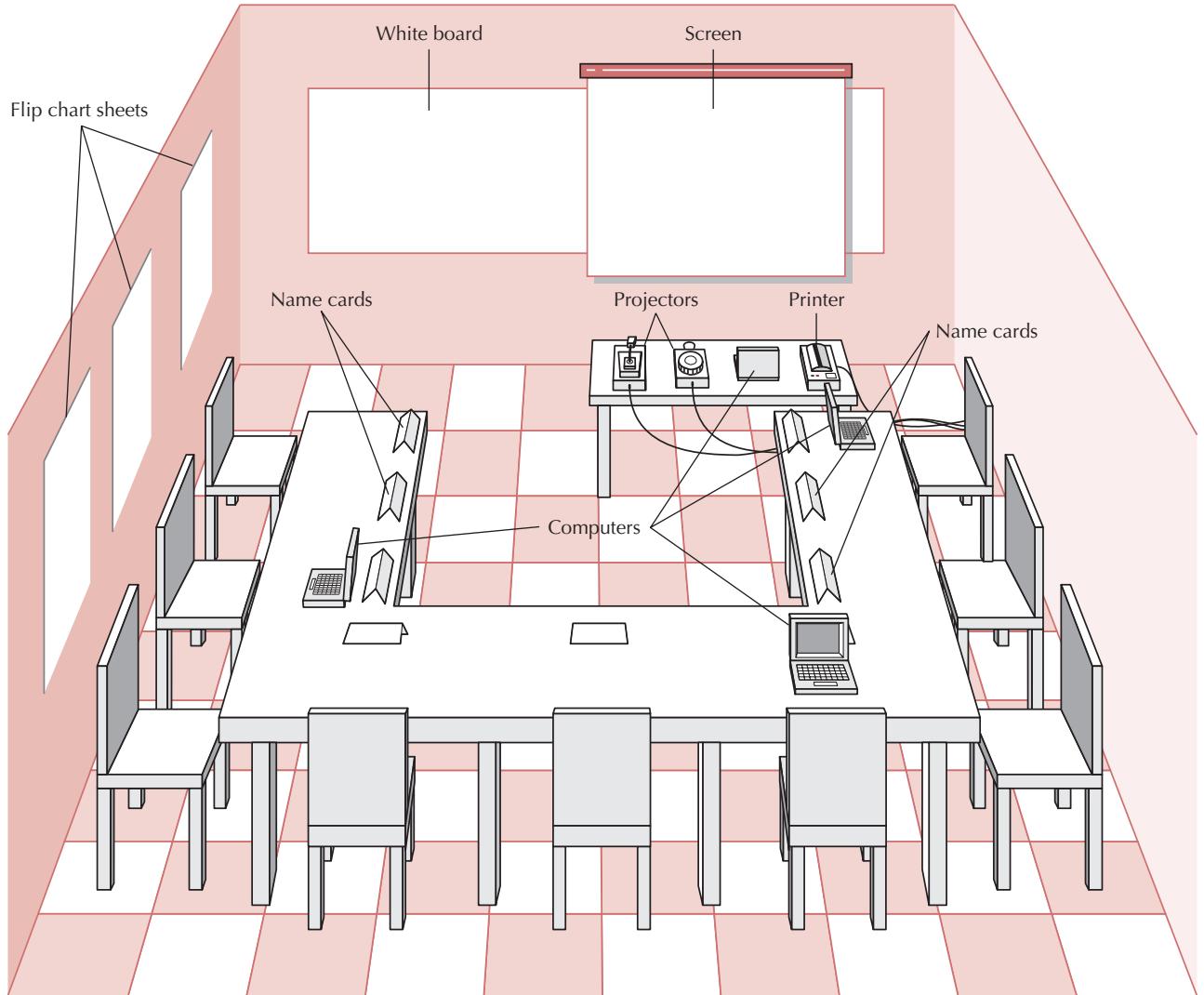


FIGURE 5-9 Joint Application Design Meeting Room

Ideally, the participants who are released from regular duties to attend the JAD sessions should be the very best people in that business unit. However, without strong management support, JAD sessions can fail because those selected to attend the JAD session are people who are less likely to be missed (i.e., the least competent people).

The facilitator should be someone who is an expert in JAD or e-JAD techniques and ideally someone who has experience with the business under discussion. In many cases, the JAD facilitator is a consultant external to the organization because the organization may not have a regular day-to-day need for JAD or e-JAD expertise. Developing and maintaining this expertise in-house can be expensive.

Designing the JAD Session JAD sessions can run from as little as a half day to several weeks, depending upon the size and scope of the project. In our experience, most JAD sessions tend to last five to ten days spread over a three-week period. Most e-JAD sessions tend to last one to four days in a one-week period. For example, the users and the analysts collectively can create analysis deliverables, such as the functional models, structural models, or the requirements definition.

As with interviewing, success depends upon a careful plan. JAD sessions usually are designed and structured using the same principles as interviews. Most JAD sessions are designed to collect specific information from users, and this requires the development of a set of questions prior to the meeting. A difference between JAD and interviewing is that all JAD sessions are structured—they *must* be carefully planned. In general, closed-ended questions are seldom used because they do not spark the open and frank discussion that is typical of JAD. In our experience, it is better to proceed top-down in JAD sessions when gathering information. Typically thirty minutes is allocated to each separate agenda item, and frequent breaks are scheduled throughout the day because participants tire easily.

Preparing for the JAD Session As with interviewing, it is important to prepare the analysts and participants for the JAD session. Because the sessions can go beyond the depth of a typical interview and are usually conducted off-site, participants can be more concerned about how to prepare. It is important that the participants understand what is expected of them. If the goal of the JAD session, for example, is to develop an understanding of the current system, then participants can bring procedure manuals and documents with them. If the goal is to identify improvements for a system, then they can think about how they would improve the system prior to the JAD Session.

Conducting the JAD Session Most JAD sessions try to follow a formal agenda, and most have formal *ground rules* that define appropriate behavior. Common ground rules include following the schedule, respecting others, opinions, accepting disagreement, and ensuring that only one person talks at a time.

The role of the JAD facilitator can be challenging. Many participants come to the JAD session with strong feelings about the system to be discussed. Channeling these feelings so that the session moves forward in a positive direction and getting participants to recognize and accept—but not necessarily agree on—opinions and situations different from their own requires significant expertise in systems analysis and design, JAD, and interpersonal skills. Few systems analysts attempt to facilitate JAD sessions without being trained in JAD techniques, and most apprentice with a skilled JAD facilitator before they attempt to lead their first session.

The JAD facilitator performs three key functions. First, he or she ensures that the group sticks to the agenda. The only reason to digress from the agenda is when it becomes clear to the facilitator, project leader, and project sponsor that the JAD session has produced some new information that is unexpected and requires the JAD session (and perhaps the project) to move in a new direction. When participants attempt to divert the discussion away from the agenda, the facilitator must be firm but polite in leading discussion back to the agenda and getting the group back on track.

Second, the facilitator must help the group understand the technical terms and jargon that surround the system development process, and help the participants understand the specific analysis techniques used. Participants are experts in their area, their part of the business, but they are not experts in systems analysis. The facilitator must therefore minimize the learning required and teach participants how to effectively provide the right information.

Third, the facilitator records the group's input on a public display area, which can be a whiteboard, flip chart, or computer display. He or she structures the information that the group provides and helps the group recognize key issues and important solutions. Under no circumstance should the facilitator insert his or her opinions into the discussion. The facilitator *must* remain neutral at all times and simply help the group through the process. The moment the facilitator offers an opinion on an issue, the group will no longer see him or her as a neutral party, but rather as someone who could be attempting to sway the group into some predetermined solution.

However, this does not mean that the facilitator should not try to help the group resolve issues. For example, if two items appear to be the same to the facilitator, the facilitator should not say, "I think these may be similar." Instead, the facilitator should ask, "Are these similar?" If the group decides they are, the facilitator can combine them and move on. However, if the group decides they are not similar (despite what the facilitator believes), the facilitator should accept the decision and move on. The group is *always* right, and the facilitator has no opinion.

Post JAD Follow-up As with interviews, a JAD postsession report is prepared and circulated among session attendees. The *postsession report* is essentially the same as the interview report in Figure 5-8. Since the JAD sessions are longer and provide more information, it usually takes a week or two after the JAD session before the report is complete.

Questionnaires

A *questionnaire* is a set of written questions for obtaining information from individuals. Questionnaires often are used when there is a large number of people from whom information and opinions are needed. In our experience, questionnaires are commonly used for systems intended for use outside of the organization (e.g., by customers or vendors) or for systems with business users spread across many geographic locations. Most people automatically think of paper when they think of questionnaires, but today more questionnaires are being distributed in electronic form, either via e-mail or on the Web. Electronic distribution can save a significant amount of money compared to distributing paper questionnaires.

Selecting Participants As with interviews and JAD sessions, the first step is to select the individuals to whom the questionnaire will be sent. However, it is not usual to select every person who could provide useful information. The standard approach is to select a *sample*, or subset, of people who are representative of the entire group. Sampling guidelines are discussed in most statistics books, and most business schools include courses that cover the topic, so we will not discuss it here. The important point in selecting a sample, however, is to realize that not everyone who receives a questionnaire will actually complete it. On average, only 30 to 50 percent of paper and e-mail questionnaires are returned. Response rates for Web-based questionnaires tend to be significantly lower (often only 5 to 30 percent).

YOUR TURN

5-5 JAD Practice

Organize yourselves into groups of four to seven people, and pick one person in each group to be the JAD facilitator. Using a blackboard, whiteboard, or flip chart, gather information about how the group performs some process

(e.g., working on a class assignment, making a sandwich, paying bills, getting to class). How did the JAD session go? Based on your experience, what are pros and cons of using JAD in a real organization?



PRACTICAL

TIP

5-2 Managing Problems in JAD Sessions

I have run more than a hundred JAD sessions and have learned several standard “facilitator tricks.” Here are some common problems and some ways to deal with them.

- **Reducing domination.** The facilitator should ensure that no one person dominates the group discussion. The only way to deal with someone who dominates is head on. During a break, approach the person, thank him or her for their insightful comments, and ask them to help you make sure that others also participate.
- **Encouraging noncontributors.** Drawing out people who have participated very little is challenging because you want to bring them into the conversation so that they will contribute again. The best approach is to ask a direct factual question that you are *certain* they can answer. And it helps to ask the question using some repetition to give them time to think. For example “Pat, I know you’ve worked shipping orders a long time. You’ve probably been in the Shipping Department longer than anyone else. Could you help us understand exactly what happens when an order is received in Shipping?”
- **Side discussions.** Sometimes participants engage in side conversations and fail to pay attention to the group. The easiest solution is simply to walk close to the people and continue to facilitate right in front of them. Few people will continue a side conversation when you are two feet from them and the entire group’s attention is on you and them.
- **Agenda merry-go-round.** The merry-go-round occurs when a group member keeps returning to the same issue every few minutes and won’t let go. One solution is to let the person have five minutes to ramble on about the issue while you carefully write down every point on a flip chart or computer file. This flip chart or file is then posted conspicuously on the wall. When the person brings up the issue again, you interrupt them, walk to the paper
- and ask them what to add. If they mention something already on the list, you quickly interrupt, point out that it is there, and ask what other information to add. Don’t let them repeat the same point, but write any new information.
- **Violent agreement.** Some of the worst disagreements occur when participants really agree on the issues but don’t realize that they agree because they are using different terms. An example is arguing whether a glass is half empty or half full; they agree on the facts, but can’t agree on the words. In this case, the facilitator has to translate the terms into different words and find common ground so the parties recognize that they really agree.
- **Unresolved conflict.** In some cases, participants don’t agree and can’t understand how to determine what alternatives are better. You can help by structuring the issue. Ask for criteria by which the group will identify a good alternative (e.g., “Suppose this idea really did improve customer service. How would I recognize the improved customer service?”). Then once you have a list of criteria, ask the group to assess the alternatives using them.
- **True conflict.** Sometimes, despite every attempt, participants just can’t agree on an issue. The solution is to postpone the discussion and move on. Document the issue as an “open issue” and list it prominently on a flip chart. Have the group return to the issue hours later. Often the issue will resolve itself by then and you haven’t wasted time on it. If the issue cannot be resolved later, move it to the list of issues to be decided by the project sponsor or some other more senior member of management.
- **Use humor.** Humor is one of the most power tools a facilitator has and thus must be used judiciously. The best JAD humor is always in context; never tell jokes but take the opportunity to find the humor in the situation.

—Alan Dennis

Designing the Questionnaire Developing good questions is critical for questionnaires because the information on a questionnaire cannot be immediately clarified for a confused respondent. Questions on questionnaires must be very clearly written and leave little room for misunderstanding, so closed-ended questions tend to be most commonly used. Questions must clearly enable the analyst to separate facts from opinions. Opinion questions often ask the respondent the extent to which they agree or disagree (e.g., “Are network problems common?”), while factual questions seek more precise values (e.g., “How often

does a network problem occur: once an hour, once a day, once a week?”). See Figure 5-10 for guidelines on questionnaire design.

Perhaps the most obvious issue—but one that is sometimes overlooked—is to have a clear understanding of how the information collected from the questionnaire will be analyzed and used. You must address this issue before you distribute the questionnaire, because it is too late afterward.

Questions should be relatively consistent in style, so that the respondent does not have to read instructions for each question before answering it. It is generally good practice to group related questions together to make them simpler to answer. Some experts suggest that questionnaires should start with questions important to respondents, so that the questionnaire immediately grabs their interest and induces them to answer it. Perhaps the most important step is to have several colleagues review the questionnaire and then pretest it with a few people drawn from the groups to whom it will be sent. It is surprising how often seemingly simple questions can be misunderstood.

Administering the questionnaire The key issue in administering the questionnaire is getting participants to complete the questionnaire and send it back. Dozens of marketing research books have been written about ways to improve response rates. Commonly used techniques include: clearly explaining why the questionnaire is being conducted and why the respondent has been selected; stating a date by which the questionnaire is to be returned; offering an inducement to complete the questionnaire (e.g., a free pen); and offering to supply a summary

YOUR TURN

5-6 Questionnaire Practice

Organize yourselves into small groups. Have each person develop a short questionnaire to collect information about the frequency in which group members perform some process (e.g., working on a class assignment, making a sandwich, paying bills, getting to class), how long it takes them, how they feel about the process, and opportunities for improving the process.

Once everyone has completed his or her questionnaire, ask each member to pass it to the right, and then complete his or her neighbor’s questionnaire. Pass the questionnaire back to the creator when it is completed.

Questions:

1. How did the questionnaire you completed differ from the one you created?
2. What are the strengths of each questionnaire?
3. How would you analyze the survey results if you had received fifty responses?
4. What would you change about the questionnaire that you developed?

- Begin with nonthreatening and interesting questions.
- Group items into logically coherent sections.
- Do not put important items at the very end of the questionnaire.
- Do not crowd a page with too many items.
- Avoid abbreviations.
- Avoid biased or suggestive items or terms.
- Number questions to avoid confusion.
- Pretest the questionnaire to identify confusing questions.
- Provide anonymity to respondents.

FIGURE 5-10
Good Questionnaire Design

of the questionnaire responses. Systems analysts have additional techniques to improve response rates inside the organization, such as personally handing out the questionnaire and personally contacting those who have not returned them after a week or two, as well as requesting the respondents' supervisors to administer the questionnaires in a group meeting.

Questionnaire Follow-up It is helpful to process the returned questionnaires and develop a questionnaire report soon after the questionnaire deadline. This ensures that the analysis process proceeds in a timely fashion and that respondents who requested copies of the results receive them promptly.

Document Analysis

Project teams often use *document analysis* to understand the as-is system. Under ideal circumstances, the project team that developed the existing system will have produced documentation, which was then updated by all subsequent projects. In this case, the project team can start by reviewing the documentation and examining the system itself.

Unfortunately, most systems are not well documented because project teams fail to document their projects along the way, and when the projects are over—there is no time to go back and document. Therefore, there may not be much technical documentation about the current systems available, or it may not contain updated information about recent system changes. However, there are many helpful documents that do exist in the organization: paper reports, memorandums, policy manuals, user training manuals, organization charts, and forms.

But these documents (forms, reports, policy manuals, organization charts) only tell part of the story. They represent the *formal system* that the organization uses. Quite often, the “real” or *informal system* differs from the formal one, and these differences, particularly large ones, give strong indications of what needs to be changed. For example, forms or reports that are never used likely should be eliminated. Likewise, boxes or questions on forms that are never filled in (or are used for other purposes) should be rethought. See Figure 5-11 for an example of how a document can be interpreted.

The most powerful indication that the system needs to be changed is when users create their own forms or add additional information to existing ones. Such changes clearly demonstrate the need for improvements to existing systems. Thus, it is useful to review both blank and completed forms to identify these deviations. Likewise, when users access multiple reports to satisfy their information needs, it is a clear sign that new information or new information formats are needed.

Observation

Observation, the act of watching processes being performed, is a powerful tool for gathering information about the as-is system because it enables the analyst to see the reality of a situation, rather than listening to others describe it in interviews or JAD sessions. Several research studies have shown that many managers really do not remember how they work and how they allocate their time. (Quick, how many hours did you spend last week on each of your courses?) Observation is a good way to check the validity of information gathered from indirect sources such as interviews and questionnaires.⁹

In many ways, the analyst becomes an anthropologist as he or she walks through the organization and observes the business system as it functions. The goal is to keep a low profile, to not interrupt those working, and to not influence those being observed. Nonetheless, it is important to understand that what analysts observe may not be the normal

⁹ A good book on observation is James P. Spradley, *Participant Observation* (New York, NY: Holt, Rinehart, and Winston, 1980).

**CENTRAL VETERINARY CLINIC
Patient Information Card**

Name:	<u>Duffy</u>	Pat Smith
Pet's Name:	Buffy	Collie 7/6/99
Address:	100 Central Court. Apartment 10	
Toronto, Ontario K7L 3N6		
Phone Number:	416-	555-3400
Do you have insurance:	yes	
Insurance Company:	Pet's Mutual	
Policy Number:	KA-5493243	

TEMPLATE
can be found at
www.wiley.com/college/dennis

The customer made a mistake. This should be labeled **Owner's Name** to prevent confusion.

The staff had to add additional information about the type of animal and the animal's date of birth. This information should be added to the new form in the to-be system.

The customer did not include area code in the phone number. This should be made more clear.

FIGURE 5-11
Performing a Document Analysis

day-to-day routine because people tend to be extremely careful in their behavior when they are being watched. Even though normal practice may be to break formal organizational rules, the observer is unlikely to see this. (Remember how you drove the last time a police car followed you?) Thus, what you see may *not* be what you get.

Observation is often used to supplement interview information. The location of a person's office and its furnishings gives clues as to their power and influence in the organization, and can be used to support or refute information given in an interview. For example, an analyst might become skeptical of someone who claims to use the existing computer system extensively if the computer is never turned on while the analyst visits. In most cases, observation will support the information that users provide in interviews. When it does not, it is an important signal that extra care must be taken in analyzing the business system.

Selecting the Appropriate Techniques

Each of the requirements-gathering techniques just discussed has strengths and weaknesses. No one technique is always better than the others, and in practice most projects use a combination of techniques. Thus, it is important to understand the strengths and weaknesses of each technique and when to use each (see Figure 5-12). One issue not discussed is that of the analysts' experience. In general, document analysis and observation require the least amount of training, while JAD sessions are the most challenging.

	Interviews	Joint Application Design	Questionnaires	Document Analysis	Observation
Type of information	As-is, improvements, to-be	As-is, improvements, to-be	As-is, improvements	As-is	As-is
Depth of information	High	High	Medium	Low	Low
Breadth of information	Low	Medium	High	High	Low
Integration of information	Low	High	Low	Low	Low
User involvement	Medium	High	Low	Low	Low
Cost	Medium	Low-Medium	Low	Low	Low-Medium

FIGURE 5-12 Table of Requirements-Gathering Techniques

CONCEPTS

5-E Publix Credit Card Forms

IN ACTION

At my neighborhood Publix grocery store, the cashiers always hand write the total amount of the charge on every credit card charge form, even though it is printed on the form. Why? Because the "back office" staff people who reconcile the cash in the cash drawers with the amount sold at the end of each shift find it hard to read the small print on the credit card forms. Writing in large print makes it easier for them to add the values up. However, cashiers sometimes make

mistakes and write the wrong amount on the forms, which causes problems.

—Barbara Wixom

Questions:

1. What does the credit card charge form indicate about the existing system?
2. How can you make improvements with a new system?

YOUR TURN

5-7 Observation Practice

Visit the library at your college or university and observe how the book check-out process occurs. First watch several students checking books out, and then check one out yourself. Prepare a brief summary report of your observations.

When you return to class, share your observations with others. You may notice that not all the reports present the same information. Why? How would the information be different had you used the interview or JAD technique?

Type of Information The first characteristic is type of information. Some techniques are more suited for use at different stages of the analysis process, whether understanding the as-is system, identifying improvements, or developing the to-be system. Interviews and JAD are commonly used in all three stages. In contrast, document analysis and observation usually are most helpful for understanding the as-is, although occasionally they provide information about current problems that need to be improved. Questionnaires are often used to gather information about the as-is system, as well as general information about improvements.

Depth of information The depth of information refers to how rich and detailed the information is that the technique usually produces, and the extent to which the technique is useful at obtaining not only facts and opinions, but also an understanding of *why* those facts and opinions exist. Interviews and JAD sessions are very useful at providing a good depth of rich and detailed information and helping the analyst to understand the reasons behind them. At the other extreme, document analysis and observation are useful for obtaining facts, but little beyond that. Questionnaires can provide a medium depth of information, soliciting both facts and opinions with little understanding of why.

Breadth of Information Breadth of information refers to the range of information and information sources that can be easily collected using that technique. Questionnaires and document analysis both are easily capable of soliciting a wide range of information from a large number of information sources. In contrast, interviews and observation require the analyst to visit each information source individually and, therefore, take more time. JAD sessions are in the middle because many information sources are brought together at the same time.

Integration of Information One of the most challenging aspects of requirements gathering is the integration of information from different sources. Simply put, different people can provide conflicting information. Combining this information and attempting to resolve differences in opinions or facts is usually very time consuming because it means contacting each information source in turn, explaining the discrepancy, and attempting to refine the information. In many cases, the individual wrongly perceives that the analyst is challenging his or her information, when in fact it is another user in the organization. This can make the user defensive and make it hard to resolve the differences.

All techniques suffer integration problems to some degree, but JAD sessions are designed to improve integration because all information is integrated when it is collected, not afterward. If two users provide conflicting information, the conflict becomes immediately obvious, as does the source of the conflict. The immediate integration of information is the single most important benefit of JAD that distinguishes it from other techniques, and this is why most organizations use JAD for important projects.

User Involvement User involvement refers to the amount of time and energy the intended users of the new system must devote to the analysis process. It is generally agreed that as users become more involved in the analysis process, the greater the chance of success. However, user involvement can have a significant cost, and not all users are willing to contribute valuable time and energy. Questionnaires, document analysis, and observation place the least burden on users, while JAD sessions require the greatest effort.

Cost Cost is always an important consideration. In general, questionnaires, document analysis, and observation are low cost techniques (although observation can be quite time consuming). The low cost does not imply that they are more or less effective than the other

techniques. We regard interviews and JAD sessions as having moderate costs. In general, JAD sessions are much more expensive initially, because they require many users to be absent from their offices for significant periods of time, and they often involve highly paid consultants. However, JAD sessions significantly reduce the time spent in information integration and thus cost less in the long term.

Combining Techniques In practice, requirements gathering combines a series of different techniques. Most analysts start by using interviews with senior manager(s) to gain an understanding of the project and the “big picture” issues. From this, the scope becomes clear as to whether large or small changes are anticipated. These are often followed with analysis of documents and policies to gain some understanding of the as-is system. Usually interviews come next to gather the rest of the information needed for the as-is system.

In our experience, identifying improvements is most commonly done using JAD sessions because the JAD session enables the users and key stakeholders to work together through an analysis technique and come to a shared understanding of the possibilities for the to-be system. Occasionally, these JAD sessions are followed by questionnaires sent to a much wider set of users or potential users to see whether the opinions of those who participated in the JAD sessions are widely shared.

Developing the concept for the to-be system is often done through interviews with senior managers, followed by JAD sessions with users of all levels to make sure the key needs of the new system are well understood.

APPLYING THE CONCEPTS AT CD SELECTIONS

Once the CD Selections approval committee approved the system proposal and feasibility analysis, the project team began performing analysis activities. These included gathering requirements using a variety of techniques, and analyzing the requirements that were gathered. An Internet marketing and sales consultant, Chris Campbell, was hired to advise Alec, Margaret, and the project team during the analysis phase. Some highlights of the project team’s activities are presented next.

Requirements Analysis Techniques

Margaret suggested that the project team conduct several JAD sessions with store managers, marketing analysts, and Web-savvy members of the IT staff. Together, the groups could work through some BPI techniques and brainstorm how improvements could be made to the current order process using a new Web-based system.

Alec facilitated three JAD sessions that were conducted over the course of a week. Alec’s past facilitation experience helped the eight-person meetings run smoothly and stay on track. First, Alec used technology analysis and suggested several important Web technologies that could be used for the system. The JAD session generated ideas about how CD Selections could apply each of the technologies to the Internet sales system project. Alec had the group categorize the ideas into three sets: “definite” ideas that would have a good probability of providing business value; “possible” ideas that might add business value; and “unlikely” ideas.

Next, Alec applied informal benchmarking by introducing the Web sites of several leading retailers and pointing out the features that they offered online. He selected some sites based on their success with Internet sales, and others based on their similarity to the vision for CD Selections’ new system. The group discussed the features that were common across most retailers versus unique functionality, and they created a list of suggested business requirements for the project team.

Requirements-Gathering Techniques

Alec believed that it would be important to understand the order processes and systems that already existed in the organization because they would have to be closely integrated with the Internet sales system. Three requirements-gathering techniques proved to be helpful in understanding the current systems and processes—document analysis, interviews, and observation.

First, the project team collected existing reports (e.g., order forms, screenshots of the online order screens) and system documentation that shed light on the as-is system. They were able to gather a good amount of information about the brick-and-mortar order processes and systems in this way. When questions arose, they conducted short interviews with the person who provided the documentation for clarification.

Next, Alec interviewed the senior analysts for the order and inventory systems to get a better understanding of how those systems worked. He asked if they had any ideas for the new system, as well as any integration issues that would need to be addressed. Alex also interviewed a contact from the ISP and the IT person who supported CD Selections' current Web site—both provided information about the existing communications infrastructure at CD Selections and its Web capabilities. Finally, Alex spent a half day visiting two of the retail stores and observing exactly how the order and hold processes worked in the brick-and-mortar facilities.

Requirements Definition

Throughout all of these activities, the project team collected information and tried to identify the business requirements for the system from the information. As the project progressed, requirements were added to the requirements definition and grouped by requirement type. When questions arose, they worked with Margaret, Chris, and Alec to confirm that requirements were in scope. The requirements that fell outside of the scope of the current system were typed into a separate document that would be saved for future use.

At the end of analysis, the requirements definition was distributed to Margaret, two marketing employees who would work with the system on the business side, and several retail store managers. This group then met for a two-day JAD session to clarify, finalize, and prioritize business requirements and to create use cases (Chapter 6) to show how the system would be used.

The project team also spent time creating structural and behavioral models (Chapters 7 and 8) that depicted the objects in the future system. Members of marketing and IT departments reviewed the documents during interviews with the project team. Figure 5-13 shows a portion of the final requirements definition.

System Proposal

Alec reviewed the requirements definition and the other deliverables that the project team created during the analysis phase. Given Margaret's desire to have the system operating before next year's Christmas season, Alec decided to timebox the project, and he determined what functionality could be included in the system by that schedule deadline (see Chapter 3). He suggested that the project team develop the system in three versions rather than attempting to develop a complete system that provided all the features initially (Phased Development, see Chapter 2). The first version, to be operational well before the holidays, would implement a “basic” system that would have the “standard” order features of other Internet retailers. The second version, planned for late spring or early summer, would have several features unique to CD Selections. The third version would add more “advanced” features, such as the ability to listen to a sample of music over the Internet, to find similar CDs, and to write reviews.

Nonfunctional Requirements

1. Operational Requirements

- 1.1 The Internet sales system will draw information from the main CD information database, which contains basic information about CDs (e.g., title, artist, ID number, price, quantity in inventory). The Internet sales system will not write information to the main CD information database.
- 1.2 The Internet sales system will store orders for new CDs in the special order system and will rely on the special order system to complete the special orders generated.
- 1.3 A new module for the in-store system will be written to manage the “holds” generated by the Internet sales system. The requirements for this new module will be documented as part of the Internet sales system because they are necessary for the Internet sales system to function.

2. Performance Requirements

No special requirements performance requirements are anticipated

3. Security Requirements

No special security requirements are anticipated

4. Cultural and Political Requirements

No special cultural and political requirements are anticipated

Functional Requirements

1. Maintain CD Information

- 1.1 The Internet sales system will need a database of basic information about the CDs that it can sell over the Internet, similar to the CD database at each of the retail stores (e.g., title, artist, id number, price, quantity in inventory).
- 1.2 Every day, the Internet sales system will receive an update from the distribution system that that will be used to update this CD database. Some new CDs will be added, some will be deleted, and others will be revised (e.g., a new price).
- 1.3 The electronic marketing (EM) manager (a position that will need to be created) will also have the ability to update information (e.g., prices for sales).

FIGURE 5-13 CD Selections Requirements Definition (Continues)

Alec revised the workplan accordingly, and he worked with Margaret and the folks in Marketing to review the feasibility analysis and update it where appropriate. All of the deliverables from the project were then combined into a system proposal and submitted to the approval committee. Figure 5-14 shows the outline of the CD Selections system proposal. Margaret and Alec met with the committee and presented the highlights of what was learned during the analysis phase and the final concept of the new system. Based on the proposal and presentation, the approval committee decided that they would continue to fund the Internet sales system.

SUMMARY

Analysis

Analysis focuses on capturing the business requirements for the system. It identifies the “what” of the system and leads directly into design, during which the “how” of the system is determined. Many deliverables are created during analysis, including the requirements definition, functional models, structural models, and behavioral models. At the end of analysis, all of these deliverables, along with revised planning and project management deliverables, are combined into a system proposal and submitted to the approval committee for a decision regarding whether to move ahead with the project.

2. Maintain CD Marketing Information

- 2.1 The Internet sales system provides an additional opportunity to market CDs to current and new customers. The system will provide a database of marketing materials about selected CDs that will help Web users learn more about them (e.g., music reviews, links to Web sites, artist information, and sample sound clips). When information about a CD that has additional marketing information is displayed, a link will be provided to the additional information.
- 2.2 Marketing materials will be supplied primarily by vendors and record labels so that we can better promote their CDs. The EM manager of the marketing department will determine what marketing materials will be placed in the system and will be responsible for adding, changing, and deleting the materials.

3. Place CD Orders

- 3.1 Customers will access the Internet sales system to look for CDs of interest. Some customers will search for specific CDs or CDs by specific artists, while other customers will want to browse for interesting CDs in certain categories (e.g., rock, jazz, classical).
- 3.2 When the customer has found all the CDs he or she wants, the customer will "check out" by providing personal information (e.g., name, e-mail, address, credit card), and information regarding the order (e.g., the CDs to purchase, and the quantity for each item).
- 3.3 The system will verify the customer's credit card information with an online credit card clearance center and either accept the order or reject it.
- 3.4 Customers will also be able check to see if their preferred stores have the CDs in stock. They will use zip code to find stores close to their location. If the CD is available at a preferred store, a customer can immediately place a hold on the CD in stock and then come into the store and pick it up.
- 3.5 If the CD is not available in the customer's preferred store, the customer can request that the CD be special ordered to that store for later pickup. The customer will be notified by e-mail when the requested CD arrives at the requested store; the CD will be placed on hold (which will again expire after 7 days). This process will work similarly to the current special order systems already available in the regular stores.
- 3.6 Alternatively, the customer can mail order the CD (see requirement 4).

4. Fill Mail Orders

- 4.1 When a CD is mail ordered, the Internet sales system will send the mail order to the mail order distribution system.
- 4.2 The mail order distribution system will handle the actual sending of CDs to customers; it will notify the Internet sales system and e-mail the customer.
- 4.3 Weekly reports can be run by the EM manager to check the order status.

FIGURE 5-13 CD Selections Requirements Definition (Continued)

Requirements Determination

Requirements determination is the part of analysis whereby the project team turns the very high-level explanation of the business requirements stated in the system request into a more precise list of requirements. A requirement is simply a statement of what the system must do or what characteristic it needs to have. Business requirements describe the "what" of the systems, and system requirements describe "how" the system will be implemented. A functional requirement relates directly to a process the system has to perform or information it needs to contain. Nonfunctional requirements refer to behavioral properties that the system must have, such as performance and usability. All of the functional and non-functional business requirements that fit within the scope of the system are written in the requirements definition, which is used to create other analysis deliverables and leads to the initial design for the new system.

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 5-14
Outline of the CD Selections System Proposal

1. Table of Contents
2. Executive Summary
A summary of all the essential information in the proposal so a busy executive can read it quickly and decide what parts of the plan to read in more depth.
3. System Request
The revised system request form (see Chapter 3).
4. Workplan
The original workplan, revised after having completed the analysis phase (see Chapter 4)
5. Requirements Definition
A list of the functional and nonfunctional business requirements for the system (this chapter).
6. Feasibility Analysis
A revised feasibility analysis, using the information from the analysis phase (see Chapter 3).
7. Functional Models
An activity diagram and a set of use cases that illustrate the basic processes that the system needs to support (see Chapter 6).
8. Structural Models
A set of structural models for the to-be system (see Chapter 7). This may include structural models of the current as-is system that will be replaced.
9. Behavioral Models
A set of behavioral models for the to-be system (see Chapter 8). This may include behavioral models of the as-is system that will be replaced.
Appendices
These contain additional material relevant to the proposal, often used to support the recommended system. This might include results of a questionnaire survey or interviews, industry reports and statistics, and so on.

Requirements Analysis Techniques

The basic process of analysis is divided into three steps: understanding the as-is system, identifying improvements, and developing requirements for the to-be system. Three requirements analysis techniques—business process automation, business process improvement, or business process reengineering—help the analyst lead users through the three (or two) analysis steps so that the vision of the system can be developed. Business process automation (BPA) means leaving the basic way in which the organization operates unchanged, and using computer technology to do some of the work. Problem analysis and root cause analysis are two popular BPA techniques. Business process improvement (BPI) means making moderate changes to the way in which the organization operates to take advantage of new opportunities offered by technology or to copy what competitors are doing. Duration analysis, activity-based costing, and information benchmarking are three popular BPI activities. Business process reengineering (BPR) means changing the fundamental way in which the organization operates. Outcome analysis, technology analysis, and activity elimination are three popular BPR activities.

Requirements-Gathering Techniques

Five techniques can be used to gather the business requirements for the proposed system: interviews, joint application development, questionnaires, document analysis, and observation. Interviews involve meeting one or more people and asking them questions. There are five basic steps to the interview process: selecting interviewees, designing interview

questions, preparing for the interview, conducting the interview, and postinterview follow-up. Joint application development (JAD) allows the project team, users, and management to work together to identify requirements for the system. Electronic JAD attempts to overcome common problems associated with groups by using groupware. A questionnaire is a set of written questions for obtaining information from individuals. Questionnaires often are used when there is a large number of people from whom information and opinions are needed. Document analysis entails reviewing the documentation and examining the system itself. It can provide insights into the formal and informal system. Observation, the act of watching processes being performed, is a powerful tool for gathering information about the as-is system because it enables the analyst to see the reality of a situation firsthand.

KEY TERMS

Activity elimination	Informal benchmarking	Questionnaire
Activity-based costing	Informal system	Requirement
Analysis	Interpersonal skill	Requirements definition
As-is system	Interview	Requirements determination
Benchmarking	Interview notes	Risk
Bottom-up interview	Interview report	Root cause
Breadth of analysis	Interview schedule	Root cause analysis
Business process automation (BPA)	Joint application development (JAD)	Sample
Business process improvement (BPI)	Nonfunctional requirements	Scribe
Business process reengineering (BPR)	Observation	Stakeholders
Business requirement	Open-ended question	Structured interview
Closed-ended question	Outcome analysis	Symptom
Critical thinking skills	Parallelization	System proposal
Document analysis	Problem analysis	System requirements
Duration analysis	Process integration	Technology analysis
Electronic JAD (e-JAD)	Postsession report	To-be system
Facilitator	Potential business value	Top-down interview
Formal system	Probing question	Unstructured interview
Functional requirement	Problem analysis	Walkthrough
Ground rule	Project cost	

QUESTIONS

- What are the key deliverables that are created during the analysis phase? What is the final deliverable from the analysis phase, and what does it contain?
- Explain the difference between an as-is system and a to-be system.
- What is the purpose of the requirements definition?
- What are the three basic steps of the analysis process? Which step is sometimes skipped or done in a cursory fashion? Why?
- Compare and contrast the business goals of BPA, BPI, and BPR.
- Compare and contrast problem analysis and root cause analysis. Under what conditions would you use problem analysis? Under what conditions would you use root cause analysis?
- Compare and contrast duration analysis and activity-based costing.
- Assuming time and money were not important concerns, would BPR projects benefit from additional time spent understanding the as-is system? Why or why not?
- What are the important factors in selecting an appropriate analysis strategy?
- Describe the five major steps in conducting interviews.
- Explain the difference between a closed-ended question, an open-ended question, and a probing question. When would you use each?

12. Explain the differences between unstructured interviews and structured interviews. When would you use each approach?
13. Explain the difference between a top-down and bottom-up interview approach. When would you use each approach?
14. How are participants selected for interviews and JAD sessions?
15. How can you differentiate between facts and opinions? Why can both be useful?
16. Describe the five major steps in conducting JAD sessions.
17. How does a JAD facilitator differ from a scribe?
18. What are the three primary things that a facilitator does in conducting the JAD session?
19. What is e-JAD, and why might a company be interested in using it?
20. How does designing questions for questionnaires differ from designing questions for interviews or JAD sessions?
21. What are typical response rates for questionnaires and how can you improve them?
22. Describe document analysis.
23. How does the formal system differ from the informal system? How does document analysis help you understand both?
24. What are the key aspects of using observation in the information-gathering process?
25. Explain factors that can be used to select information-gathering techniques

EXERCISES

- A. Review the Amazon.com Web site. Develop the requirements definition for the site. Create a list of functional business requirements that the system meets. What different kinds of nonfunctional business requirements does the system meet? Provide examples for each kind.
- B. Pretend that you are going to build a new system that automates or improves the interview process for the Career Services Department of your school. Develop a requirements definition for the new system. Include both functional and nonfunctional system requirements. Pretend you will release the system in three different versions. Prioritize the requirements accordingly.
- C. Describe in very general terms the as-is business process for registering for classes at your university. What BPA technique would you use to identify improvements? With whom would you use the BPA technique? What requirements-gathering technique would help you apply the BPA technique? List some example improvements that would you expect to find.
- D. Describe in very general terms the as-is business process for registering for classes at your university. What BPI technique would you use to identify improvements? With whom would you use the BPI technique? What requirements-gathering technique would help you apply the BPI technique? List some example improvements that you would expect to find.
- E. Describe in very general terms the as-is business process for registering for classes at your university. What BPR technique would you use to identify improvements? With whom would you use the BPR technique? What requirements-gathering technique
- would help you apply the BPR technique? List some example improvements that would you expect to find.
- F. Suppose your university is having a dramatic increase in enrollment and is having difficulty finding enough seats in courses for students so they can take courses required for graduation. Perform a technology analysis to identify new ways to help students complete their studies and graduate.
- G. Suppose you are the analyst charged with developing a new system for the university bookstore with which students can order books online and have them delivered to their dorms and off-campus housing. What requirements-gathering techniques will you use? Describe in detail how you would apply the techniques.
- H. Suppose you are the analyst charged with developing a new system to help senior managers make better strategic decisions. What requirements-gathering techniques will you use? Describe in detail how you would apply the techniques.
- I. Find a partner and interview each other about what tasks you/they did in the last job held (full-time, part-time, past or current). If you haven't worked before, then assume your job is being a student. Before you do this, develop a brief interview plan. After your partner interviews you, identify the type of interview, interview approach, and types of questions used.
- J. Find a group of students and run a sixty-minute JAD session on improving alumni relations at your university. Develop a brief JAD plan, select two techniques that will help identify improvements, and then develop an agenda. Conduct the session using the agenda, and write your postsession report.

- K. Find a questionnaire on the Web that has been created to capture customer information. Describe the purpose of the survey, the way questions are worded, and how the questions have been organized. How can it be improved? How will the responses be analyzed?
- L. Develop a questionnaire that will help gather information regarding processes at a popular restaurant, or the college cafeteria (e.g., ordering, customer service).

Give the questionnaire to ten to fifteen students, analyze the responses, and write a brief report that describes the results.

- M. Contact the Career Services Department at your university and find all the pertinent documents designed to help students find permanent and/or part-time jobs. Analyze the documents and write a brief report.

MINICASES

- The State Firefighter's Association has a membership of 15,000. The purpose of the organization is to provide some financial support to the families of deceased member firefighters and to organize a conference each year bringing together firefighters from all over the state. Annually members are billed dues and calls. "Calls" are additional funds required to take care of payments made to the families of deceased members. The bookkeeping work for the association is handled by the elected treasurer, Bob Smith, although it is widely known that his wife, Laura, does all of the work. Bob runs unopposed each year at the election, since no one wants to take over the tedious and time-consuming job of tracking memberships. Bob is paid a stipend of \$8,000 per year, but his wife spends well over twenty hours per week on the job. The organization however, is not happy with their performance.

A computer system is used to track the billing and receipt of funds. This system was developed in 1984 by a Computer Science student and his father. The system is a DOS-based system written using dBase 3. The most immediate problem facing the treasurer and his wife is the fact that the software package no longer exists, and there is no one around who knows how to maintain the system. One query in particular takes seventeen hours to run. Over the years, they have just avoided running this query, although the information in it would be quite useful. Questions from members concerning their statements cannot be easily answered. Usually Bob or Laura just jot down the inquiry and return a call with the answer. Sometimes it takes three to five hours to find the information needed to answer the question. Often, they have to perform calculations manually since the system was not programmed to handle certain types of queries. When member information is entered into the system, each field is presented one at a time. This makes it very difficult to return to a field and correct a value that was entered. Sometimes a new

member is entered but disappears from the records. The report of membership used in the conference materials does not alphabetize members by city. Only cities are listed in the correct order.

What requirements analysis strategy or strategies would you recommend for this situation? Explain your answer.

- Brian Callahan, IS Project Manager, is just about ready to depart for an urgent meeting called by Joe Campbell, Manager of Manufacturing Operations. A major BPI project sponsored by Joe recently cleared the approval hurdle, and Brian helped bring the project through project initiation. Now that the approval committee has given the go-ahead, Brian has been working on the project's analysis plan.

One evening, while playing golf with a friend who works in the Manufacturing Operations Department, Brian learned that Joe wants to push the project's time frame up from Brian's original estimate of thirteen months. Brian's friend overheard Joe say, "I can't see why that IS project team needs to spend all that time 'analyzing' things. They've got two weeks scheduled just to look at the existing system! That seems like a real waste. I want that team to get going on building my system."

Because Brian has a little inside knowledge about Joe's agenda for this meeting, he has been considering how to handle Joe. What do you suggest Brian tell Joe?

- Barry has recently been assigned to a project team that will be developing a new retail store management system for a chain of submarine sandwich shops. Barry has several years of experience in programming but has not done much analysis in his career. He was a little nervous about the new work he would be doing, but was confident he could handle any assignment he was given.

One of Barry's first assignments was to visit one of the submarine sandwich shops and prepare an observation report on how the store operates. Barry

planned to arrive at the store around noon, but he chose a store in an area of town he was unfamiliar with, and due to traffic delays and difficulty in finding the store, he did not arrive until 1:30 PM. The store manager was not expecting him and refused to let a stranger behind the counter until Barry had him contact the project sponsor (the Director of Store Management) back at company headquarters to verify who he was and what his purpose was.

After finally securing permission to observe, Barry stationed himself prominently in the work area behind the counter so that he could see everything. The staff had to maneuver around him as they went about their tasks, and there were only minor occasional collisions. Barry noticed that the store staff seemed to be going about their work very slowly and deliberately, but he supposed that was because the store wasn't very busy. At first, Barry questioned each worker about what he or she was doing, but the store manager eventually asked him not to interrupt their work so much—he was interfering with their service to the customers.

By 3:30 PM, Barry was a little bored. He decided to leave, figuring he could get back to the office and prepare his report before 5:00 PM that day. He was sure his team leader would be pleased with his quick completion of his assignment. As he drove, he reflected, "There really won't be much to say in this report. All they do is take the order, make the sandwich, collect the payment, and hand over the order. It's really simple!" Barry's confidence in his analytical skills soared as he anticipated his team leader's praise.

Back at the store, the store manager shook his head, commenting to his staff, "He comes here at the slowest time of day on the slowest day of the week. He never even looked at all the work I was doing in the back room while he was here—summarizing yesterday's sales,

checking inventory on hand, making up resupply orders for the weekend ... plus he never even considered our store opening and closing procedures. I hate to think that the new store management system is going to be built by someone like that. I'd better contact Chuck (the Director of Store Management) and let him know what went on here today." Evaluate Barry's conduct of the observation assignment.

4. Anne has been given the task of conducting a survey of sales clerks who will be using a new order entry system being developed for a household products catalog company. The goal of the survey is to identify the clerks' opinions on the strengths and weaknesses of the current system. There are about fifty clerks who work in three different cities, so a survey seemed like an ideal way of gathering the needed information from the clerks.

Anne developed the questionnaire carefully and pretested it on several sales supervisors who were available at corporate headquarters. After revising it based on their suggestions, she sent a paper version of the questionnaire to each clerk, asking that it be returned within one week. After one week, she had only three completed questionnaires returned. After another week, Anne received just two more completed questionnaires. Feeling somewhat desperate, Anne then sent out an e-mail version of the questionnaire, again to all the clerks, asking them to respond to the questionnaire by e-mail as soon as possible. She received two e-mail questionnaires and three messages from clerks who had completed the paper version expressing annoyance at being bothered with the same questionnaire a second time. At this point, Anne has just a 14 percent response rate, which she is sure will not please her team leader. What suggestions do you have that could have improved Anne's response rate to the questionnaire?

PART TWO

ANALYSIS PHASE

The Analysis Phase answers the questions of *who* will use the system, *what* the system will do, and *where* and *when* it will be used. During this phase, the project team will learn about the system. The team then produces the Functional Model (Activity Diagrams, Use Case Descriptions and Diagram), Structural Model (CRC Cards and Class and Object Diagrams), and Behavioral Models (Sequence Diagrams, Communication Diagrams, and Behavioral State Machines).

CHAPTER 6 □FUNCTIONAL
MODELING**CHAPTER 7 □**STRUCTURAL
MODELING**CHAPTER 8 □**BEHAVIORAL
MODELING

CHAPTER 5

REQUIREMENTS DETERMINATION

One of the first activities of an analyst is to determine the business requirements for the system. This chapter begins by presenting the requirements definition, a document that lists the new system's capabilities. It then describes how to analyze requirements using business process automation, business process improvement, and business process reengineering techniques, and how to gather requirements using interviews, JAD sessions, questionnaires, document analysis, and observation.

OBJECTIVES

- Understand how to create a requirements definition.
- Become familiar with requirements analysis techniques.
- Understand when to use each requirements analysis technique.
- Understand how to gather requirements using interviews, JAD sessions, questionnaires, document analysis, and observation.
- Understand when to use each requirements-gathering technique.

CHAPTER OUTLINE

Introduction	Interviews
Requirements Determination	Joint Application Development
What is a Requirement?	Questionnaires
Requirements Definition	Document Analysis
Determining Requirements	Observation
Creating the Requirements Definition	Selecting the Appropriate Techniques
Requirements Analysis Techniques	Applying the Concepts at CD Selections
Business Process Automation	Requirements Analysis Techniques
Business Process Improvement	Requirements-Gathering Techniques
Business Process Reengineering	Requirements Definition
Selecting the Appropriate Technique	System Proposal
Requirement-Gathering Techniques	Summary

INTRODUCTION

The systems development life cycle (SDLC) is the process by which the organization moves from the current system (often called the *as-is system*) to the new system (often called the *to-be system*). The output of planning, discussed in Chapters 3 and 4, is the system request, which provides general ideas for the to-be system, defines the project's scope, and provides

the initial workplan. The analysis phase takes the general ideas in the system request and refines them into a detailed requirements definition (this chapter), functional models (Chapter 6), structural models (Chapter 7), and behavioral models (Chapter 8) that together form the system proposal. The *system proposal* also includes revised project management deliverables, such as the feasibility analysis (Chapter 3) and the workplan (Chapter 4).

The system proposal is presented to the approval committee, who decides if the project is to continue. This usually happens at a system *walkthrough*, a meeting at which the concept for the new system is presented to the users, managers, and key decision makers. The goal of the walkthrough is to explain the system in moderate detail so that the users, managers, and key decision makers clearly understand it, can identify needed improvements, and are able to make a decision about whether the project should continue. If approved, the system proposal moves into the design phase, and its elements (requirements definition, functional, structural, and behavioral models) are used as inputs to the steps in design. This further refines them and defines in much more detail how the system will be built.

The line between analysis and design is very blurry. This is because the deliverables created during analysis are really the first step in the design of the new system. Many of the major design decisions for the new system are found in the analysis deliverables. In fact, a better name for analysis would really be “analysis and initial design,” but because this is a rather long name, and because most organizations simply call it analysis, we will too. Nonetheless, it is important to remember that the deliverables from analysis are really the first step in the design of the new system.

In many ways, the requirements determination step is the single most critical step of the entire SDLC, because it is here that the major elements of the system first begin to emerge. During requirements determination, the system is easy to change because little work has been done yet. As the system moves through the other phases in the SDLC, it becomes harder and harder to return to requirements determination and to make major changes because of all of the rework that is involved. Several studies have shown that more than half of all system failures are due to problems with the requirements.¹ This is why the iterative approaches of many object-oriented methodologies are so effective—small batches of requirements can be identified and implemented in incremental stages, allowing the overall system to evolve over time.

In this chapter, we focus on the requirements-determination step of analysis. We begin by explaining what a requirement is and the overall process of requirements gathering and requirements analysis. We then present a set of techniques that can be used to analyze and gather requirements.

REQUIREMENTS DETERMINATION

The purpose of the *requirements determination* step is to turn the very high-level explanation of the business requirements stated in the system request into a more precise list of requirements than can be used as inputs to the rest of analysis (creating functional, structural, and behavioral models). This expansion of the requirements ultimately leads to the design phase.

What is a Requirement?

A *requirement* is simply a statement of what the system must do or what characteristic it must have. During analysis, requirements are written from the perspective of the businessperson, and they focus on the “what” of the system. They focus on business user needs, so they usually are called *business requirements* (and sometimes user requirements). Later

¹ For example, see *The Scope of Software Development Project Failures* by The Standish Group, Dennis MA, 1995.

in design, business requirements evolve to become more technical, and they describe “how” the system will be implemented. Requirements in design are written from the developer’s perspective, and they usually are called *system requirements*.

Before we continue, we want to stress that there is no black-and-white line dividing a business requirement and a system requirement—and some companies use the terms interchangeably. The important thing to remember is that a requirement is a statement of what the system must do, and requirements will change over time as the project moves from analysis to design to implementation. Requirements evolve from detailed statements of the business capabilities that a system should have to detailed statements of the technical way in which the capabilities will be implemented in the new system.

Requirements can be either functional or nonfunctional in nature. A *functional requirement* relates directly to a process the system has to perform or information it needs to contain. For example, requirements that state that the system must have the ability to search for available inventory or to report actual and budgeted expenses are functional requirements. Functional requirements flow directly into the next steps of analysis (functional, structural, and behavioral models) because they define the functions that the system needs to have.

Nonfunctional requirements refer to behavioral properties that the system must have, such as performance and usability. The ability to access the system using a Web browser would be considered a nonfunctional requirement. Nonfunctional requirements may influence the rest of analysis (functional, structural, and behavioral models) but often do so only indirectly; nonfunctional requirements are primarily used in design when decisions are made about the user interface, the hardware and software, and the system’s underlying physical architecture.

Figure 5-1 lists different kinds of nonfunctional requirements and examples of each kind. Notice that the nonfunctional requirements describe a variety of characteristics regarding the system: operational, performance, security, and cultural and political. These characteristics do not describe business processes or information, but they are very important in understanding what the final system should be like. For example, the project team

YOUR TURN

5-1 Identifying Requirements

One of the most common mistakes by new analysts is to confuse functional and nonfunctional requirements. Pretend that you received the following list of requirements for a sales system.

Requirements for Proposed System:
The system should...

1. Be accessible to Web users
2. Include the company standard logo and color scheme
3. Restrict access to profitability information
4. Include actual and budgeted cost information
5. Provide management reports
6. Include sales information that is updated at least daily

7. Have 2-second maximum response time for predefined queries, and 10-minute maximum response time for ad hoc queries
8. Include information from all company subsidiaries
9. Print subsidiary reports in the primary language of the subsidiary
10. Provide monthly rankings of salesperson performance

Questions:

1. Which requirements are functional business requirements? Provide two additional examples.
2. Which requirements are nonfunctional business requirements? What kind of nonfunctional requirements are they? Provide two additional examples.

Nonfunctional Requirement	Description	Examples
Operational	The physical and technical environments in which the system will operate	<ul style="list-style-type: none"> ■ The system should be able to fit in a pocket or purse ■ The system should be able to integrate with the existing inventory system ■ The system should be able to work on any Web browser
Performance	The speed, capacity, and reliability of the system	<ul style="list-style-type: none"> ■ Any interaction between the user and the system should not exceed 2 seconds ■ The system should receive updated inventory information every 15 minutes ■ The system should be available for use 24 hours per day, 365 days per year
Security	Who has authorized access to the system under what circumstances	<ul style="list-style-type: none"> ■ Only direct managers can see personnel records of staff ■ Customers can only see their order history during business hours
Cultural and Political	Cultural, political factors and legal requirements that affect the system	<ul style="list-style-type: none"> ■ The system should be able to distinguish between United States and European currency ■ Company policy says that we only buy computers from Dell ■ Country managers are permitted to authorize customer user interfaces within their units ■ The system shall comply with insurance industry standards

Source: The Atlantic Systems Guild, <http://www.systemsguild.com>

FIGURE 5-1
Nonfunctional Requirements

CONCEPTS

IN ACTION

5-A What Can Happen If You Ignore Nonfunctional Requirements

I once worked on a consulting project in which my manager created a requirements definition without listing non-functional requirements. The project was then estimated based on the requirements definition and sold to the client for \$5,000. In my manager's mind, the system that we would build for the client would be a very simple stand-alone system running on current technology. It shouldn't take more than a week to analyze, design, and build.

Unfortunately, the client had other ideas. They wanted the system to be used by many people in three different departments, and they wanted the ability for any number of people to work on the system concurrently. The technology they had in place was antiquated, but

nonetheless they wanted the system to run effectively on the existing equipment. Because we didn't set the project scope properly by including our assumptions about non-functional requirements in the requirements definition, we basically had to do whatever they wanted.

The capabilities they wanted took weeks to design and program. The project ended up taking four months, and the final project cost was \$250,000. Our company had to pick up the tab for everything except the agreed upon \$5,000. This was by far the most frustrating project situation I ever experienced.

—Barbara Wixom

needs to know if a system needs to be highly secure, requires subsecond response time, or has to reach a multilingual customer base. These requirements will impact design decisions that will be made in design, particularly physical architecture design, so we will revisit them in detail in Chapter 13. The goal at this point is to identify any major issues.

Requirements Definition

The requirements definition report—usually just called the *requirements definition*—is a straightforward text report that simply lists the functional and nonfunctional requirements in an outline format. Figure 5-2 shows a sample requirements definition for a word processing program designed to compete against software, such as Microsoft Word.

The requirements are numbered in a legal or outline format so that each requirement is clearly identified. The requirements are first grouped into functional and nonfunctional requirements and then within each of those headings they are further grouped by the type of nonfunctional requirement or by function.

TEMPLATE
can be found at
[www.wiley.com/
college/dennis](http://www.wiley.com/college/dennis)

D. Nonfunctional Requirements

1. Operational Requirements

- 1.1. The system will operate in Windows and Macintosh environments
- 1.2. The system will be able to read and write Word documents, RTF, and HTML
- 1.3. The system will be able to import Gif, Jpeg, and BMP graphics files

2. Performance Requirements

- 2.1. Response times must be less than 7 seconds
- 2.2. The Inventory database must be updated in real time

3. Security Requirements

- 3.1. No special security requirements are anticipated

4. Cultural and Political Requirements

- 4.1. No special cultural and political requirements are anticipated

C. Functional Requirements

1. Printing

- 1.1. The user can select which pages to print
- 1.2. The user can view a preview of the pages before printing
- 1.3. The user can change the margins, paper size (e.g., letter, A4) and orientation on the page

2. Spell Checking

- 2.1. The user can check for spelling mistakes; the system can operate in one of two modes as selected by the users
 - 2.1.1. Mode 1 (Manual): The user will activate the spell checker and it will move the user to the next misspelled word
 - 2.1.2. Mode 2 (Automatic): As the user types, the spell checker will flag misspelled words so the user immediately see the misspelling
- 2.2. The user can add words to the dictionary
- 2.3. The user can mark words as not misspelled but not add them to the dictionary

FIGURE 5-2
Sample Requirements
Definition

Sometimes, business requirements are prioritized on the requirements definition. They can be ranked as having high, medium, or low importance in the new system, or they can be labeled with the version of the system that will address the requirement (e.g., release 1, release 2, release 3). This practice is particularly important when using object-oriented methodologies since they deliver requirements in batches by developing incremental versions of the system.

The most *obvious* purpose of the requirements definition is to provide the information needed by the other deliverables in analysis, which include functional, structural, and behavioral models, and to support activities in the design phase. The most *important* purpose of the requirements definition, however, is to define the scope of the system. The document describes to the analysts exactly what the system needs to end up doing. When discrepancies arise, the document serves as the place to go for clarification.

Determining Requirements

Determining requirements for the requirements definition is both a business task and an IT task. In the early days of computing, there was a presumption that the systems analysts, as experts with computer systems, were in the best position to define how a computer system should operate. Many systems failed because they did not adequately address the true business needs of the users. Gradually, the presumption changed so that the users, as the business experts, were seen as being the best position to define how a computer system should operate. However, many systems failed to deliver performance benefits because users simply automated an existing inefficient system, and they failed to incorporate new opportunities offered by technology.

A good analogy is building a house or an apartment. We have all lived in a house or apartment, and most of us have some understanding of what we would like to see in one. However, if we were asked to design one from scratch, it would be a challenge because we lack appropriate design skills and technical engineering skills. Likewise, an architect acting alone would probably miss some of our unique requirements.

Therefore, the most effective approach is to have both businesspeople and analysts working together to determine business requirements. Sometimes, however, users don't know exactly what they want, and analysts need to help them discover their needs. Three kinds of techniques have become popular to help analysts do this: business process automation (BPA), business process improvement (BPI), and business process reengineering (BPR). These techniques are tools that analysts can use when they need to guide the users in explaining what is wanted from a system.

The three kinds of techniques work similarly. They help users critically examine the current state of systems and processes (the *as-is* system), identify exactly what needs to change, and develop a concept for a new system (the *to-be* system). A different amount of change is associated with each technique; BPA creates a small amount of change, BPI creates a moderate amount of change, and BPR creates significant change that affects much of the organization. All three will be described in greater detail later in the chapter.

Although BPA, BPI, and BPR enable the analyst to help users create a vision for the new system, they are not sufficient for extracting information about the detailed business requirements that are needed to build it. Therefore, analysts use a portfolio of requirement-gathering techniques to acquire information from users. The analyst has many gathering techniques from which to choose: interviews, questionnaires, observation, joint application development (JAD), and document analysis. The information gathered using these techniques is critically analyzed and used to craft the requirements definition report. The final section of this chapter describes each of the requirements-gathering techniques in greater depth.

Creating the Requirements Definition

Creating the requirements definition is an iterative and ongoing process whereby the analyst collects information with requirements-gathering techniques (e.g., interviews, document analysis), critically analyzes the information to identify appropriate business requirements for the system, and adds the requirements to the requirements definition report. The requirements definition is kept up to date so that the project team and business users can refer to it and get a clear understanding of the new system.

To create the requirements definition, the project team first determines the kinds of functional and nonfunctional requirements that they will collect about the system (of course, these may change over time). These become the main sections of the document. Next, the analysts use a variety of requirement-gathering techniques (e.g., interviews, observation) to collect information, and they list the business requirements that were identified from that information. Finally, the analysts work with the entire project team and the business users to verify, change, and complete the list and to help prioritize the importance of the requirements that were identified.

This process continues throughout analysis, and the requirements evolves over time as new requirements are identified and as the project moves into later phases of the SDLC. Beware: the evolution of the requirements definition must be carefully managed. The project team cannot keep adding to the requirements definition, or the system will keep growing and growing and never get finished. Instead, the project team carefully identifies requirements and evaluates which ones fit within the scope of the system. When a requirement reflects a real business need but is not within the scope of the current system or current release, it is added on a list of future requirements, or given a low priority. The management of requirements (and system scope) is one of the hardest parts of managing a project!

REQUIREMENTS ANALYSIS TECHNIQUES

Before the project team can determine what requirements are appropriate for a given system, they need to have a clear vision of the kind of system that will be created, and the level of change that it will bring to the organization. The basic process of *analysis* is divided into three steps: understanding the as-is system, identifying improvements, and developing requirements for the to-be system.

Sometimes the first step (i.e., understanding the as-is system) is skipped or done in a cursory manner. This happens when no current system exists, if the existing system and processes are irrelevant to the future system, or if the project team is using a RAD or agile development methodology in which the as-is system is not emphasized. Users of traditional design methods such as waterfall and parallel development (see Chapter 1) typically spend significant time understanding the as-is system and identifying improvements before moving to capture requirements for the to-be system. However, newer RAD, agile, and object-oriented methodologies, such as phased development, prototyping, throwaway prototyping, and extreme programming (see Chapters 1 and 2) focus almost exclusively on improvements and the to-be system requirements, and they spend little time investigating the current as-is system.

Three requirements analysis techniques—business process automation, business process improvement, or business process reengineering—help the analyst lead users through the three (or two) analysis steps so that the vision of the system can be developed. We should note that requirements analysis techniques and requirements-gathering techniques go hand in hand. Analysts need to use requirements-gathering techniques to collect information; requirements analysis techniques drive the kind of information that is gathered and how it is ultimately analyzed. Although we focus now on the analysis techniques

and then discuss requirements gathering at the end of the chapter, they happen concurrently and are complementary activities.

The choice of analysis technique used is based on the amount of change the system is meant to create in the organization. BPA is based on small change that improves process efficiency, BPI creates process improvements that lead to better effectiveness, and BPR revamps the way things work so that the organization is transformed on some level.

To move the users “from here to there,” an analyst needs strong *critical thinking skills*. Critical thinking is the ability to recognize strengths and weaknesses and recast an idea in an improved form, and they are needed to really understand issues and develop new business processes. These skills are needed to thoroughly examine the results of requirements gathering, to identify business requirements, and to translate those requirements into a concept for the new system.

Business Process Automation

Business process automation (BPA) means leaving the basic way in which the organization operates unchanged, and using computer technology to do some of the work. BPA can make the organization more efficient but has the least impact on the business. BPA projects spend a significant time understanding the current as-is system before moving on to improvements and to-be system requirements. Problem analysis and root cause analysis are two popular BPA techniques.

Problem Analysis The most straightforward (and probably the most commonly used) requirements analysis technique is *problem analysis*. Problem analysis means asking the users and managers to identify problems with the as-is system and to describe how to solve them in the to-be system. Most users have a very good idea of the changes they would like to see, and most will be quite vocal about suggesting them. Most changes tend to solve problems rather than capitalize on opportunities, but this is possible, too. Improvements from problem analysis tend to be small and incremental (e.g., provide more space in which to type the customer’s address; provide a new report that currently does not exist).

This type of improvement often is very effective at improving a system’s efficiency or ease of use. However, it often provides only minor improvements in business value—the new system is better than the old, but it may be hard to identify significant monetary benefits from the new system.

Root Cause Analysis The ideas produced by problem analysis tend to be *solutions* to problems. All solutions make assumptions about the nature of the problem, assumptions that may or may not be valid. In our experience, users (and most people in general) tend to jump quickly to solutions without fully considering the nature of the problem. Sometimes the solutions are appropriate, but many times they address a *symptom* of the problem, not the true problem or *root cause* itself.²

For example, suppose you notice that a lightbulb is burned out above your front door. You buy a new bulb, get out a ladder, and replace the bulb. A month later, you see that the same bulb is burnt out, so you buy a new bulb, haul out the ladder, and replace it again. This repeats itself several times. At this point, you have two choices. You can buy a large package of lightbulbs and a fancy lightbulb changer on a long pole so you don’t need the

² Some excellent books that address the importance of gathering requirements and various techniques include: Alan M. Davis, *Software Requirements: Objects, Functions, & States, Revision* (Englewood Cliffs, NJ: Prentice Hall, 1993); Gerald Kotonya and Ian Sommerville, *Requirements Engineering* (Chichester, England: John Wiley & Sons, 1998); and Dean Leffingwell and Don Widrig, *Managing Software Requirements: A Unified Approach* (Reading, MA: Addison-Wesley, 2000).

haul the ladder out each time (thus saving a lot of trips to the store for new bulbs and a lot of effort in working with the ladder). Or you can fix the light fixture that is causing the light to burn out in the first place. Buying the bulb changer is treating the symptom (the burnt-out bulb), while fixing the fixture is treating the root cause.

In the business world, the challenge lies in identifying the root cause—few problems are as simple as the lightbulb problem. The solutions that users propose (or systems that analysts think of) may either address symptoms or root causes, but without a careful analysis, it is difficult to tell which one. And finding out that you've just spent a million dollars on a new lightbulb changer is a horrible feeling!

Root cause analysis therefore focuses on problems, not solutions. The analyst starts by having the users generate a list of problems with the current system, and then prioritize the problems in order of importance. Then starting with the most important, the users and/or the analysts generate all the possible root causes for the problems. Each possible root cause is investigated (starting with the most likely or easiest to check) until the true root cause(s) are identified. If any possible root causes are identified for several problems, those should be investigated first, because there is a good chance they are the real root causes influencing the symptom problems.

In our lightbulb example, there are several possible root causes. A decision tree sometimes helps with the analysis. As Figure 5-3 shows, there are many possible root causes, so buying a new fixture may or may not address the true root cause. In fact, buying a lightbulb changer may actually address the root cause. The key point in root cause analysis is to always challenge the obvious.

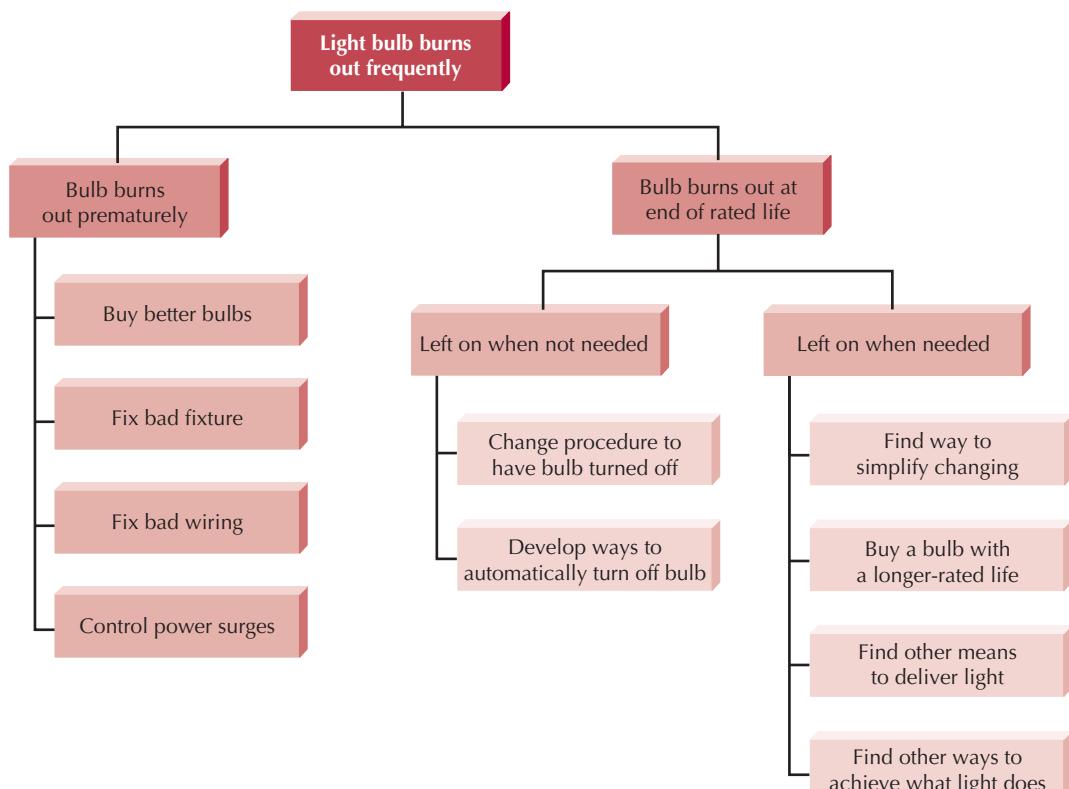


FIGURE 5-3 Root Cause Analysis for the Example of the Burned-Out Lightbulb

Business Process Improvement

Business process improvement (BPI) means making moderate changes to the way in which the organization operates to take advantage of new opportunities offered by technology or to copy what competitors are doing. BPI can improve efficiency (i.e., doing things right) and improve effectiveness (i.e., doing the right things). BPI projects also spend time understanding the as-is system, but much less time than BPA projects; their primary focus is on improving business processes, so time is spent on the as-is only to help with the improvement analyses and the to-be system requirements. Duration analysis, activity-based costing, and information benchmarking are three popular BPI activities.

Duration Analysis *Duration analysis* requires a detailed examination of the amount of time it takes to perform each process in the current as-is system. The analysts begin by determining the total amount of time it takes, on average, to perform a set of business processes for a typical input. They then time each of the individual steps (or subprocesses) in the business process. The time to complete the basic steps are then totaled and compared to the total for the overall process. When there is a significant difference between the two—and in our experiences the total time often can be 10 or even 100 times longer than the sum of the parts—this indicates that this part of the process is badly in need of a major overhaul.

For example, suppose that the analysts are working on a home mortgage system and discover that on average, it takes thirty days for the bank to approve a mortgage. They then look at each of the basic steps in the process (e.g., data entry, credit check, title search, appraisal, etc.) and find that the total amount of time actually spent on each mortgage is

CONCEPTS

IN ACTION

5-B When Optimal Isn't

In 1984, I developed an information system to help schedule production orders in paper mills. Paper is made in huge rolls that are six to ten feet wide. Customer orders (e.g., newspaper, computer paper) are cut from these large rolls. The goal of production scheduling is to decide which orders to combine on one roll to reduce the amount of paper wasted because no matter how you place the orders on the rolls, you almost always end up throwing away the last few inches—customer orders seldom add up to exactly the same width as the roll itself (imagine trying to cut several rolls of toilet paper from a paper towel roll).

The scheduling system was designed to run on an IBM PC, which in those days was not powerful enough to use advanced mathematical techniques like linear programming to calculate which orders to combine on which roll. Instead, we designed the system to use the rules of thumb that the production schedulers themselves had developed over many years. After several months of fine-tuning, the system worked very well. The schedulers were happy and the amount of waste decreased.

By 1986, PCs had increased in power, so we revised the system to use the advanced linear programming tech-

niques to provide better schedules and reduce the waste even more. In our tests, the new system reduced waste by a few percentage points over the original system, so it was installed. However, after several months, there were big problems; the schedulers were unhappy and the amount of waste actually grew.

It turned out that although the new system really did produce better schedules with less waste, the real problem was *not* figuring how to place the orders on the rolls to reduce waste. Instead, the real problem was finding orders for the next week that could be produced with the current batch. The old system combined orders in a way in which the schedulers found it easy to find “matching” future orders. The new system made odd combinations with which the schedulers had a hard time working. The old system was eventually re-installed.

—Alan Dennis

Question

1. How could the problem with the new system have been foreseen?

about eight hours. This is a strong indication that the overall process is badly broken, because it takes thirty days to perform one day's work.

These problems likely occur because the process is badly fragmented. Many different people must perform different activities before the process finishes. In the mortgage example, the application probably sits on many peoples' desks for long periods of time before it is processed. Processes in which many different people work on small parts of the inputs are prime candidates for *process integration* or *parallelization*. Process integration means changing the fundamental process so that fewer people work on the input, which often requires changing the processes and retraining staff to perform a wider range of duties. Process parallelization means changing the process so that all the individual steps are performed at the same time. For example in the mortgage application example, there is probably no reason that the credit check cannot be performed at the same time as the appraisal and title check.

Activity-Based Costing *Activity-based costing* is a similar analysis that examines the cost of each major process or step in a business process rather than the time taken.³ The analysts identify the costs associated with each of the basic functional steps or processes, identify the most costly processes, and focus their improvement efforts on them.

Assigning costs is conceptually simple. You just examine the direct cost of labor and materials for each input. Materials costs are easily assigned in a manufacturing process, while labor costs are usually calculated based on the amount of time spent on the input and the hourly cost of the staff. However, as you may recall from a managerial accounting course, there are indirect costs such as rent, depreciation, and so on that also can be included in activity costs.

Informal Benchmarking *Benchmarking* refers to studying how other organizations perform a business process in order to learn how your organization can do something better. Benchmarking helps the organization by introducing ideas that employees may never have considered, but have the potential to add value.

Informal benchmarking is fairly common for "customer-facing" business processes (i.e., those processes that interact with the customer). With informal benchmarking, the

CONCEPTS

5-C Duration Analysis

IN ACTION

A group of executives from a *Fortune* 500 company used Duration Analysis to discuss their procurement process. Using a huge wall of Velcro and a handful of placards, a facilitator proceeded to map out the company's process for procuring a \$50 software upgrade. Having quantified the time it took to complete each step, she then assigned costs based on the salaries of the employees involved. The

15-minute exercise left the group stunned. Their procurement process had gotten so convoluted that it took eighteen days, countless hours of paperwork and nearly \$22,000 in people time to get the product ordered, received, and up and running on the requester's desktop.

Source: "For Good Measure," *CIO Magazine*, March 1 1999, by Debby Young.

³ For more information on activity-based costing, see K.B. Burk and D. W. Webster, *Activity Based Costing* (Fairfax, VA: American Management Systems, 1994); and D. T. Hicks, *Activity-Based Costing: Making It Work for Small and Mid-Sized Companies* (New York: John Wiley, 1998).

managers and analysts think about other organizations, or visit them as customers to watch how the business process is performed. In many cases, the business studied may be a known leader in the industry or simply a related firm. For example, suppose the team is developing a Web site for a car dealer. The project sponsor, key managers, and key team members would likely visit the Web sites of competitors, as well as those of others in the car industry (e.g., manufacturers, accessories suppliers) and those in other industries that have won awards for their Web sites.

Business Process Reengineering

Business process reengineering (BPR) means changing the fundamental way in which the organization operates—“obliterating” the current way of doing business and making major changes to take advantage of new ideas and new technology. BPR projects spend little time understanding the *as-is*, because their goal is to focus on new ideas and new ways of doing business. Outcome analysis, technology analysis, and activity elimination are three popular BPR activities.

Outcome Analysis *Outcome analysis* focuses on understanding the fundamental outcomes that provide value to customers. While these outcomes sound as though they should be obvious, they often aren’t. For example, suppose you are an insurance company, and one of your customers has just had a car accident. What is the fundamental outcome from the *customer’s* perspective? Traditionally, insurance companies have answered this question by assuming the customer wants to receive the insurance payment quickly. To the customer, however, the payment is only a *means* to the real outcome: a repaired car. The insurance company might benefit by extending their view of the business process past its traditional boundaries to include not paying for repairs, but performing the repairs or contracting with an authorized body shop to do them.

With this approach, the system analysts encourage the managers and project sponsor to pretend they are customers and to think carefully about what the organization’s products and services enable the customers to do—and what they *could* enable the customer to do.

Technology Analysis Many major changes in business over the past decade have been enabled by new technologies. *Technology analysis* therefore starts by having the analysts and managers develop a list of important and interesting technologies. Then the group systematically identifies how each and every technology could be applied to the business process and identifies how the business would benefit.

For example, one useful technology might be the Internet. Saturn, the car manufacturer, took this idea and developed an Extranet application for its suppliers. Rather than ordering parts for its cars, Saturn makes its production schedule available electronically to its suppliers, who ship the parts Saturn needs so that they arrive at the plant just in time. This saves Saturn significant costs because it eliminates the need for people to monitor the production schedule and issue purchase orders.

Activity Elimination *Activity elimination* is exactly what it sounds like. The analysts and managers work together to identify how the organization could eliminate each and every activity in the business process, how the function could operate without it, and what effects are likely to occur. Initially, managers are reluctant to conclude that processes can be eliminated, but this is a “force-fit” exercise in that they must eliminate each activity. In some cases the results are silly, but nonetheless, participants must address each and every activity in the business process.

For example, in the home mortgage approval process discussed earlier, the managers and analysts would start by eliminating the first activity, entering the data into the mortgage company's computer. This leads to two obvious possibilities (1) eliminate the use of a computer system; or (2) make someone else do the data entry (e.g., the customer over the Web). They would then eliminate the next activity, the credit check. Silly right? After all, making sure the applicant has good credit is critical in issuing a loan. Not really. The real answer depends upon how many times the credit check identifies bad applications. If all or almost all applicants have good credit and are seldom turned down by a credit check, then the cost of the credit check may not be worth the cost of the few bad loans it prevents. Eliminating it may actually result in lower costs even with the cost of bad loans.

Selecting the Appropriate Technique

Each of the techniques discussed in this chapter has its own strengths and weaknesses (see Figure 5-4). No one technique is inherently better than the others, and in practice most projects use a combination of techniques.

Potential Business Value The *potential business value* varies with analysis strategy. While BPA has the potential to improve the business, most of the benefits from BPA are tactical and small in nature. Since BPA does not seek to change the business processes, it can only improve their efficiency. BPI usually offers moderate potential benefits, depending upon the scope of the project, because it seeks to change the business in some way. It can increase both efficiency and effectiveness. BPR creates large *potential* benefits because it seeks to radically improve the nature of the business.

Project Cost *Project cost* is always important. In general, BPA requires the lowest cost because it has the narrowest focus and seeks to make the fewest number of changes. BPI can be moderately expensive, depending upon the scope of the project. BPR is usually expensive, both because of the amount of time required of senior managers and the amount of redesign to business processes.

Breadth of Analysis *Breadth of analysis* refers to the scope of analysis, or whether analysis includes business processes within a single business function, processes that cross the organization, or processes that interact with those in customer or supplier organizations. BPR takes a broad perspective, often spanning several major business processes, even across multiple organizations. BPI has a much narrower scope that usually includes one or several business functions. BPA typically examines a single process.

Risk One final issue is *risk* of failure, which is the likelihood of failure due to poor design, unmet needs, or too much change for the organization to handle. BPA and BPI have low to moderate risk because the to-be system is fairly well defined and understood,

	Business Process Automation	Business Process Improvement	Business Process Reengineering
Potential business value	Low–moderate	Moderate	High
Project cost	Low	Low–moderate	High
Breadth of analysis	Narrow	Narrow–moderate	Very broad
Risk	Low–moderate	Low–moderate	Very high

FIGURE 5-4
Characteristics of Analysis Strategies

and its potential impact on the business can be assessed before it is implemented. BPR projects, on the other hand, are less predictable. BPR is extremely risky and not something to be undertaken unless the organization and its senior leadership are committed to making significant changes. Mike Hammer, the father of BPR, estimates that 70 percent of BPR projects fail.

YOUR TURN

5-2 IBM Credit

IBM Credit was a wholly owned subsidiary of IBM responsible for financing mainframe computers sold by IBM. While some customers bought mainframes outright, or obtained financing from other sources, financing computers provided significant additional profit.

When an IBM sales representative made a sale, he or she would immediately call IBM Credit to obtain a financing quote. The call was received by a credit officer who would record the information on a request form. The form would then be sent to the credit department to check the customer's credit status. This information would be recorded on the form, which was then sent to the business practices department who would write a contract (sometimes reflecting changes requested by the customer). The form and the contract would then go to the pricing department, which used the credit information to establish an interest rate and recorded it on the Form. The Form and contract was then sent to the clerical group, where an administrator would prepare a cover letter quoting the interest rate and send the letter and contract via Federal Express to the customer.

The problem at IBM Credit was a major one. Getting a financing quote took anywhere for four to eight days (six days on average), giving the customer time to rethink the order or find financing elsewhere. While the quote was being prepared, sales representatives would often call to find out where the quote was in the process, so they could tell the customer when to expect it. However, no one at IBM Credit could answer the question because the paper forms could be in any department and it was impossible to locate one without physically walking

through the departments and going through the piles of forms on everyone's desk.

IBM Credit examined the process and changed it so that each credit request was logged into a computer system so that each department could record an application's status as they completed it and sent it to the next department. In this way, sales representatives could call the credit office and quickly learn the status of each application. IBM used some sophisticated management science queuing theory analysis to balance workloads and staff across the different departments so none would be overloaded. They also introduced performance standards for each department (e.g., the pricing decision had to be completed within one day after that department received an application).

However, process times got worse, even though each department was achieving almost 100 percent compliance on its performance goals. After some investigation, managers found that when people got busy, they conveniently found errors that forced them to return credit requests to the previous department for correction, thereby removing it from their time measurements.

Questions:

1. What techniques can you use to identify improvements? Choose one technique and apply it to this situation—what improvements did you identify?

Source: *Reengineering the Corporation*, New York: Harper Business, 1993, by M. Hammer and J. Champy.

YOUR TURN

5-3 Analysis Strategy

Suppose you are the analyst charged with developing a new Web site for a local car dealer who wants to be very

innovative and try new things. What analysis techniques would you recommend? Why?

REQUIREMENTS-GATHERING TECHNIQUES

An analyst is very much like a detective (and business users sometimes are like elusive suspects). He or she knows that there is a problem to be solved and therefore must look for clues that uncover the solution. Unfortunately, the clues are not always obvious (and often missed), so the analyst needs to notice details, talk with witnesses, and follow leads just as Sherlock Holmes would have done. The best analysts will thoroughly gather requirements using a variety of techniques and make sure that the current business processes and the needs for the new system are well understood before moving into design. You don't want to discover later that you have key requirements wrong—surprises like this late in the SDLC can cause all kinds of problems.

The requirements-gathering process is used for building political support for the project and establishing trust and rapport between the project team building the system and the users who ultimately will choose to use or not use the system. Involving someone in the process implies that the project teams views that person as an important resource and values his or her opinions. You *must* include all of the key *stakeholders* (the people who can affect the system or who will be affected by the system) in the requirements-gathering process. This might include managers, employees, staff members, and even some customers and suppliers. If you do not involve a key person, that individual may feel slighted, which can cause problems during implementation (e.g., "How could they have developed the system without my input?!").

The second challenge of requirements gathering is choosing the way(s) in which information is collected. There are many techniques for gathering requirements that vary from asking people questions to watching them work. In this chapter, we focus on the five most commonly used techniques: interviews, JAD sessions (a special type of group meeting), questionnaires, document analysis, and observation. Each technique has its own strengths and weaknesses, many of which are complementary, so most projects use a combination of techniques, probably most often interviews, JAD sessions, and document analysis.⁴

Interviews

The *interview* is the most commonly used requirements-gathering technique. After all, it is natural—usually if you need to know something, you ask someone. In general, interviews are conducted one-on-one (one interviewer and one interviewee), but sometimes, due to time constraints, several people are interviewed at the same time. There are five basic steps to the interview process: selecting interviewees, designing interview questions, preparing for the interview, conducting the interview, and postinterview follow-up.⁵

Selecting Interviewees The first step to interviewing is to create an *interview schedule* that lists all of the people who will be interviewed, when, and for what purpose (see Figure 5-5). The schedule can be an informal list that is used to help set up meeting times, or a formal list that is incorporated into the workplan. The people who appear on the interview schedule are selected based on the analyst's information needs. The project sponsor, key business users, and other members of the project team can help the analyst determine who in the organization can best provide important information about requirements. These people are listed on the interview schedule in the order in which they should be interviewed.

⁴ Two good books that discuss the problems in finding the root causes to problems are: E.M. Goldratt, and J. Cox, *The Goal* (Croton-on-Hudson, NY: North River Press, 1986), and E.M. Goldratt, *The Haystack Syndrome* (Croton-on-Hudson, NY: North River Press, 1990).

⁵ Two good books on interviewing are Brian James, *The Systems Analysis Interview* (Manchester: NCC Blackwell, 1989); and James P. Spradley, *The Ethnographic Interview* (New York, NY: Holt, Rinehart, and Winston, 1979).

FIGURE 5-5
Sample Interview Schedule

Name	Position	Purpose of Interview	Meeting
Andria McClellan	Director, Accounting	Strategic vision for new accounting system	Mon, March 1 8:00–10:00 AM
Jennifer Draper	Manager, Accounts Receivable	Current problems with accounts receivable process; future goals	Mon, March 1 2:00–3:15 PM
Mark Goodin	Manager, Accounts Payable	Current problems with accounts payable process; future goals	Mon, March 1 4:00–5:15 PM
Anne Asher	Supervisor, Data Entry	Accounts receivable and payable processes	Wed, March 3 10:00–11:00 AM
Fernando Merce	Data Entry Clerk	Accounts receivable and payable processes	Wed, March 3 1:00–3:00 PM

**CONCEPTS
IN ACTION**

5-D Selecting the Wrong People

In 1990, I led a consulting team for a major development project for the U.S. Army. The goal was to replace eight existing systems used on virtually every Army base across the United States. The as-is analysis models for these systems had been built, and our job was to identify improvement opportunities and develop to-be process models for each of the eight systems.

For the first system, we selected a group of mid-level managers (captains and majors) recommended by their commanders as being the experts in the system under construction. These individuals were the first and second line

managers of the business function. The individuals were expert at managing the process, but did not know the exact details of how the process worked. The resulting to-be analysis models were very general and non-specific.

—Alan Dennis

Question

1. Suppose you were in charge of the project. Create an interview schedule for the remaining seven projects.

People at different levels of the organization will have different perspectives on the system, so it is important to include both managers who manage the processes and staff who actually perform the processes to gain both high-level and low-level perspectives on an issue. Also, the kinds of interview subjects that you need may change over time. For example, at the start of the project, the analyst has a limited understanding of the as-is business process. It is common to begin by interviewing one or two senior managers to get a strategic view, and then move to mid-level managers who can provide broad, overarching information about the business process and the expected role of the system being developed. Once the analyst has a good understanding of the “big picture,” lower-level managers and staff members can fill in the exact details of how the process works. Like most other things about systems analysis, this is an iterative process—starting with senior managers, moving to mid-level managers, then staff members, back to mid-level managers, and so on, depending upon what information is needed along the way.

It is quite common for the list of interviewees to grow, often by 50 percent to 75 percent. As you interview people, you likely will identify more information that is needed and additional people who can provide the information.

Designing Interview Questions There are three types of interview questions: closed-ended questions, open-ended questions, and probing questions. *Closed-ended questions* are those that require a specific answer. You can think of them as being similar to multiple choice or arithmetic questions on an exam (see Figure 5-6). Closed-ended questions are used when the analyst is looking for specific, precise information (e.g., how many credit card requests are received per day). In general, precise questions are best. For example, rather than asking “Do you handle a lot of requests?” it is better to ask “How many requests do you process per day?”

Closed-ended questions enable analysts to control the interview and obtain the information they need. However, these types of questions don’t uncover *why* the answer is the way it is, nor do they uncover information that the interviewer does not think to ask ahead of time.

Open-ended questions are those that leave room for elaboration on the part of the interviewee. They are similar in many ways to essay questions that you might find on an exam (see Figure 5-6 for examples). Open-ended questions are designed to gather rich information and give the interviewee more control over the information that is revealed during the interview. Sometimes the information that the interviewee chooses to discuss uncovers information that is just as important as the answer (e.g., if the interviewee talks only about other departments when asked for problems, it may suggest that he or she is reluctant to admit his or her own problems).

The third type of question is the *probing question*. Probing questions follow-up on what has just been discussed in order to learn more, and they often are used when the interviewer is unclear about an interviewee’s answer. They encourage the interviewee to expand on or to confirm information from a previous response, and they are a signal that the interviewer is listening and interested in the topic under discussion. Many beginning analysts are reluctant to use probing questions because they are afraid that the interviewee might be offended at being challenged or because they believe it shows that they didn’t understand what the interviewee said. When done politely, probing questions can be a powerful tool in requirements gathering.

In general, you should not ask questions about information that is readily available from other sources. For example, rather than asking what information is used to perform a task, it is simpler to show the interviewee a form or report (see document analysis later) and ask what information on it is used. This helps focus the interviewee on the task, and saves time, because he or she does not need to describe the information detail—he or she just needs to point it out on the form or report.

Types of Questions	Examples
Closed-Ended Questions	<ul style="list-style-type: none"> • How many telephone orders are received per day? • How do customers place orders? • What information is missing from the monthly sales report?
Open-Ended Questions	<ul style="list-style-type: none"> • What do you think about the current system? • What are some of the problems you face on a daily basis? • What are some of the improvements you would like to see in a new system?
Probing Questions	<ul style="list-style-type: none"> • Why? • Can you give me an example? • Can you explain that in a bit more detail?

FIGURE 5-6
Three Types of
Questions

No question type is better than another, and usually a combination of questions is used during an interview. At the initial stage of an IS development project, the as-is process can be unclear, so the interview process begins with *unstructured interviews*, interviews that seek a broad and roughly defined set of information. In this case, the interviewer has a general sense of the information needed, but few close-ended questions to ask. These are the most challenging interviews to conduct because they require the interviewer to ask open-ended questions and probe for important information “on the fly.”

As the project progresses, the analyst comes to understand the business process much better, and he or she needs very specific information about how business processes are performed (e.g., exactly how a customer credit card is approved). At this time, the analyst conducts *structured interviews*, in which specific sets of questions are developed prior to the interviews. There usually are more close-ended questions in a structured interview than in the unstructured approach.

No matter what kind of interview is being conducted, interview questions must be organized into a logical sequence, so that the interview flows well. For example, when trying to gather information about the current business process, it can be useful to move in logical order through the process or from the most important issues to the least important.

There are two fundamental approaches to organizing the interview questions: top-down or bottom-up; see Figure 5-7. With the *top-down interview*, the interviewer starts with broad, general issues and gradually works towards more specific ones. With the *bottom-up interview*, the interviewer starts with very specific questions and moves to broad questions. In practice, analysts mix the two approaches, starting with broad general issues, moving to specific questions, and then back to general issues.

The top-down approach is an appropriate strategy for most interviews (it is certainly the most common approach). The top-down approach enables the interviewee to become accustomed to the topic before he or she needs to provide specifics. It also enables the interviewer to understand the issues before moving to the details because the interviewer may not have sufficient information at the start of the interview to ask very specific questions. Perhaps most importantly, the top-down approach enables the interviewee to raise a set of “big picture” issues before becoming enmeshed in details, so the interviewer is less likely to miss important issues.

One case in which the bottom-up strategy may be preferred is when the analyst already has gathered a lot of information about issues and just needs to fill in some holes with

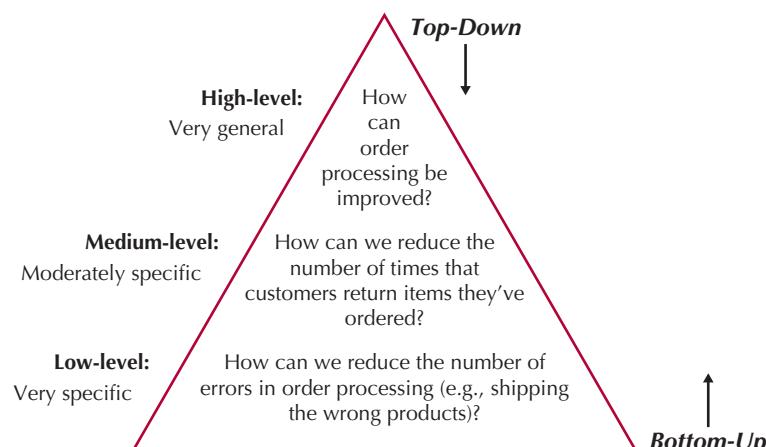


FIGURE 5-7
Top-Down and Bottom-Up Questioning Strategies

details. Or, bottom-up may be appropriate if lower-level staff members are threatened or unable to answer high-level questions. For example, “How can we improve customer service?” may be too broad a question for a customer service clerk, whereas a specific question is readily answerable (e.g., “How can we speed up customer returns?”). In any event, all interviews should begin with noncontroversial questions first, and then gradually move into more contentious issues after the interviewer has developed some rapport with the interviewee.

Preparing for the Interview It is important to prepare for the interview in the same way that you would prepare to give a presentation. You should have a general interview plan that lists the questions that you will ask in the appropriate order; anticipates possible answers and provides how you will follow up with them; and identifies segues between related topics. Confirm the areas in which the interviewee has knowledge so you do not ask questions that he or she cannot answer. Review the topic areas, the questions, and the interview plan, and clearly decide which have the greatest priority in case you run out of time.

In general, structured interviews with closed-ended questions take more time to prepare than unstructured interviews. So, some beginning analysts prefer unstructured interviews, thinking that they can “wing it.” This is very dangerous and often counterproductive, because any information not gathered in the first interview would require follow-up efforts, and most users do not like to be interviewed repeatedly about the same issues.

PRACTICAL

TIP

5-1 Developing Interpersonal Skills



Interpersonal skills are those skills than enable you to develop rapport with others, and they are very important for interviewing. They help you to communicate with others effectively. Some people develop good interpersonal skills at an early age; they simply seem to know how to communicate and interact with others. Other people are less “lucky” and need to work hard to develop their skills. Interpersonal skills, like most skills, can be learned. Here are some tips:

- **Don’t worry, be happy.** Happy people radiate confidence and project their feelings on others. Try interviewing someone while smiling and then interviewing someone else while frowning and see what happens!
- **Pay attention.** Pay attention to what the other person is saying (which is harder than you might think). See how many times you catch yourself with your mind on something other than the conversation at hand.
- **Summarize key points.** At the end of each major theme or idea that someone explains, you should

repeat the key points back to the speaker (e.g., “Let me make sure I understand. The key issues are...”). This demonstrates that you consider the information important—and also forces you to pay attention (you can’t repeat what you didn’t hear).

- **Be succinct.** When you speak, be succinct. The goal in interviewing (and in much of life) is to learn, not to impress. The more you speak, the less time you give to others.
- **Be honest.** Answer all questions truthfully, and if you don’t know the answer, say so.
- **Watch body language (yours and theirs).** The way a person sits or stands conveys much information. In general, a person who is interested in what you are saying sits or leans forward, makes eye contact, and often touches his or her face. A person leaning away from you or with an arm over the back of a chair is disinterested. Crossed arms indicate defensiveness or uncertainty, while “steepling” (sitting with hands raised in front of the body with fingertips touching) indicates a feeling of superiority.

Be sure to prepare the interviewee as well. When you schedule the interview, inform the interviewee of the reason for the interview and the areas you will be discussing far enough in advance so that he or she has time to think about the issues and organize his or her thoughts. This is particularly important when you are an outsider to the organization, and for lower-level employees who often are not asked for their opinions and who may be uncertain about why you are interviewing them.

Conducting the Interview When you start the interview, the first goal is to build rapport with the interviewee, so that he or she trusts you and is willing to tell you the whole truth, not just give the answers that he or she thinks you want. You should appear to be professional and an unbiased, independent seeker of information. The interview should start with an explanation of why you are there and why you have chosen to interview the person, and then move into your planned interview questions.

It is critical to carefully record all the information that the interviewee provides. In our experience, the best approach is to take careful notes—write down *everything* the interviewee says, even if it does not appear immediately relevant. Don't be afraid to ask the person to slow down or to pause while you write, because this is a clear indication that the interviewee's information is important to you. One potentially controversial issue is whether or not to tape-record the interview. Recording ensures that you do not miss important points, but it can be intimidating for the interviewee. Most organizations have policies or generally accepted practices about the recording of interviews, so find out what they are before you start an interview. If you are worried about missing information and cannot tape the interview, then bring along a second person to take detailed notes.

As the interview progresses, it is important that you understand the issues that are discussed. If you do not understand something, be sure to ask. Don't be afraid to ask "dumb questions" because the only thing worse than appearing "dumb" is to be "dumb" by not understanding something. If you don't understand something during the interview, you certainly won't understand it afterward. Try to recognize and define jargon, and be sure to clarify jargon you do not understand. One good strategy to increase your understanding during an interview is to periodically summarize the key points that the interviewee is communicating. This avoids misunderstandings and also demonstrates that you are listening.

Finally, be sure to separate facts from opinion. The interviewee may say, for example, "we process too many credit card requests." This is an opinion, and it is useful to follow this up with a probing question requesting support for the statement (e.g., "Oh, how many do you process in a day?"). It is helpful to check the facts because any differences between the facts and the interviewee's opinions can point out key areas for improvement. Suppose the interviewee complains about a high or increasing number of errors, but the logs show that errors have been decreasing. This suggests that errors are viewed as a very important problem that should be addressed by the new system, even if they are declining.

As the interview draws to a close, be sure to give the interviewee time to ask questions or provide information that he or she thinks is important but was not part of your interview plan. In most cases, the interviewee will have no additional concerns or information, but in some cases this will lead to unanticipated, but important information. Likewise, it can be useful to ask the interviewee if there are other people who should be interviewed. Make sure that the interview ends on time (if necessary, omit some topics or plan to schedule another interview).

As a last step in the interview, briefly explain what will happen next (see the next section). You don't want to prematurely promise certain features in the new system or a specific delivery date, but you do want to reassure the interviewee that his or her time was well spent and very helpful to the project.

Post-Interview Follow-up After the interview is over, the analyst needs to prepare an *interview report* that describes the information from the interview (Figure 5-8). The report contains *interview notes*, information that was collected over the course of the interview and is summarized in a useful format. In general, the interview report should be written within forty-eight hours of the interview, because the longer you wait, the more likely you are to forget information.

Often, the interview report is sent to the interviewee with a request to read it and inform the analyst of clarifications or updates. Make sure the interviewee is convinced that you genuinely want his or her corrections to the report. Usually there are few changes, but the need for any significant changes suggests that a second interview will be required. Never distribute someone's information without prior approval.

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 5-8
Interview Report

Interview Notes Approved By: Linda Estey
<p>Person Interviewed: Linda Estey, Director, Human Resources</p> <p>Interviewer: Barbara Wixom</p> <p>Purpose of Interview:</p> <ul style="list-style-type: none"> • Understand reports produced for Human Resources by the current system • Determine information requirements for future system <p>Summary of Interview:</p> <ul style="list-style-type: none"> • Sample reports of all current HR reports are attached to this report. The information that is not used and missing information are noted on the reports. • Two biggest problems with the current system are: <ol style="list-style-type: none"> 1. The data is too old (the HR Department needs information within two days of month end; currently information is provided to them after a three-week delay) 2. The data is of poor quality (often reports must be reconciled with departmental HR database) • The most common data errors found in the current system include incorrect job level information and missing salary information. <p>Open Items:</p> <ul style="list-style-type: none"> • Get current employee roster report from Mary Skudrna (extension 4355). • Verify calculations used to determine vacation time with Mary Skudrna. • Schedule interview with Jim Wack (extension 2337) regarding the reasons for data quality problems. <p>Detailed Notes: See attached transcript.</p>

YOUR TURN

5-4 Interview Practice

Interviewing is not as simple as it first appears. Select two people from class to go to the front of the room to demonstrate an interview. (This also can be done in groups.) Have one person be the interviewer, and the other the interviewee. The interviewer should conduct a 5-minute interview regarding the school course registration system. Gather information about the existing system and how the system can be improved. If there is time, repeat with another pair.

Questions:

1. Describe the body language of the interview pair.
2. What kind of interview was conducted?
3. What kinds of questions were asked?
4. What was done well? How could the interview be improved?

Joint Application Development(JAD)

Joint application development (or *JAD* as it is more commonly known) is an information gathering technique that allows the project team, users, and management to work together to identify requirements for the system. IBM developed the *JAD* technique in the late 1970s, and it is often the most useful method for collecting information from users.⁶ Capers Jones claims that *JAD* can reduce scope creep by 50 percent, and it avoids the requirements for a system from being too specific or too vague, both of which cause trouble during later stages of the *SDLC*.⁷ *JAD* is a structured process in which ten to twenty users meet together under the direction of *facilitator* skilled in *JAD* techniques. The facilitator is a person who sets the meeting agenda and guides the discussion, but does not join in the discussion as a participant. He or she does not provide ideas or opinions on the topics under discussion to remain neutral during the session. The facilitator must be an expert in both group process techniques and systems analysis and design techniques. One or two *scribes* assist the facilitator by recording notes, making copies, and so on. Often the scribes will use computers and *CASE* tools to record information as the *JAD* session proceeds.

The *JAD* group meets for several hours, several days, or several weeks until all of the issues have been discussed and the needed information is collected. Most *JAD* sessions take place in a specially prepared meeting room, away from the participants' offices so that they are not interrupted. The meeting room is usually arranged in a U shape so that all participants can easily see each other (see Figure 5-9). At the front of the room (the open part of the "U"), there is a whiteboard, flip chart and/or overhead projector for use by the facilitator who leads the discussion.

One problem with *JAD* is that it suffers from the traditional problems associated with groups; sometimes people are reluctant to challenge the opinions of others (particularly their boss), a few people often dominate the discussion, and not everyone participates. In a fifteen-member group, for example, if everyone participates equally, then each person can talk for only four minutes each hour and must listen for the remaining fifty-six minutes—not a very efficient way to collect information.

A new form of *JAD* called *electronic JAD* or *e-JAD* attempts to overcome these problems by using groupware. In an *e-JAD* meeting room, each participant uses special software on a networked computer to send anonymous ideas and opinions to everyone else. In this way, all participants can contribute at the same time, without fear of reprisal from people with differing opinions. Initial research suggests that *e-JAD* can reduce the time required to run *JAD* sessions by 50 percent to 80 percent.⁸

Selecting Participants Selecting *JAD* participants is done in the same basic way as selecting interview participants. Participants are selected based on the information they can contribute, to provide a broad mix of organizational levels, and to build political support for the new system. The need for all *JAD* participants to be away from their office at the same time can be a major problem. The office may need to be closed or run with a "skeleton" staff until the *JAD* sessions are complete.

⁶ More information on *JAD* can be found in J. Wood and D. Silver, *Joint Application Development* (New York: John Wiley & Sons, 1989); and Alan Cline, "Joint Application Development for Requirements Collection and Management," <http://www.carolla.com/wp-jad.htm>.

⁷ See Kevin Strehlo, "Catching up with the Jones and 'Requirement' Creep," *InfoWorld*, July 29, 1996, and Kevin Strehlo, "The Makings of a Happy Customer: Specifying Project X" *InfoWorld*. Nov 11, 1996.

⁸ For more information on *e-JAD*, see A. R. Dennis, G. S. Hayes, and R. M. Daniels, "Business Process Modeling with Groupware," *Journal of MIS* 15(4), 1999, 115–142.

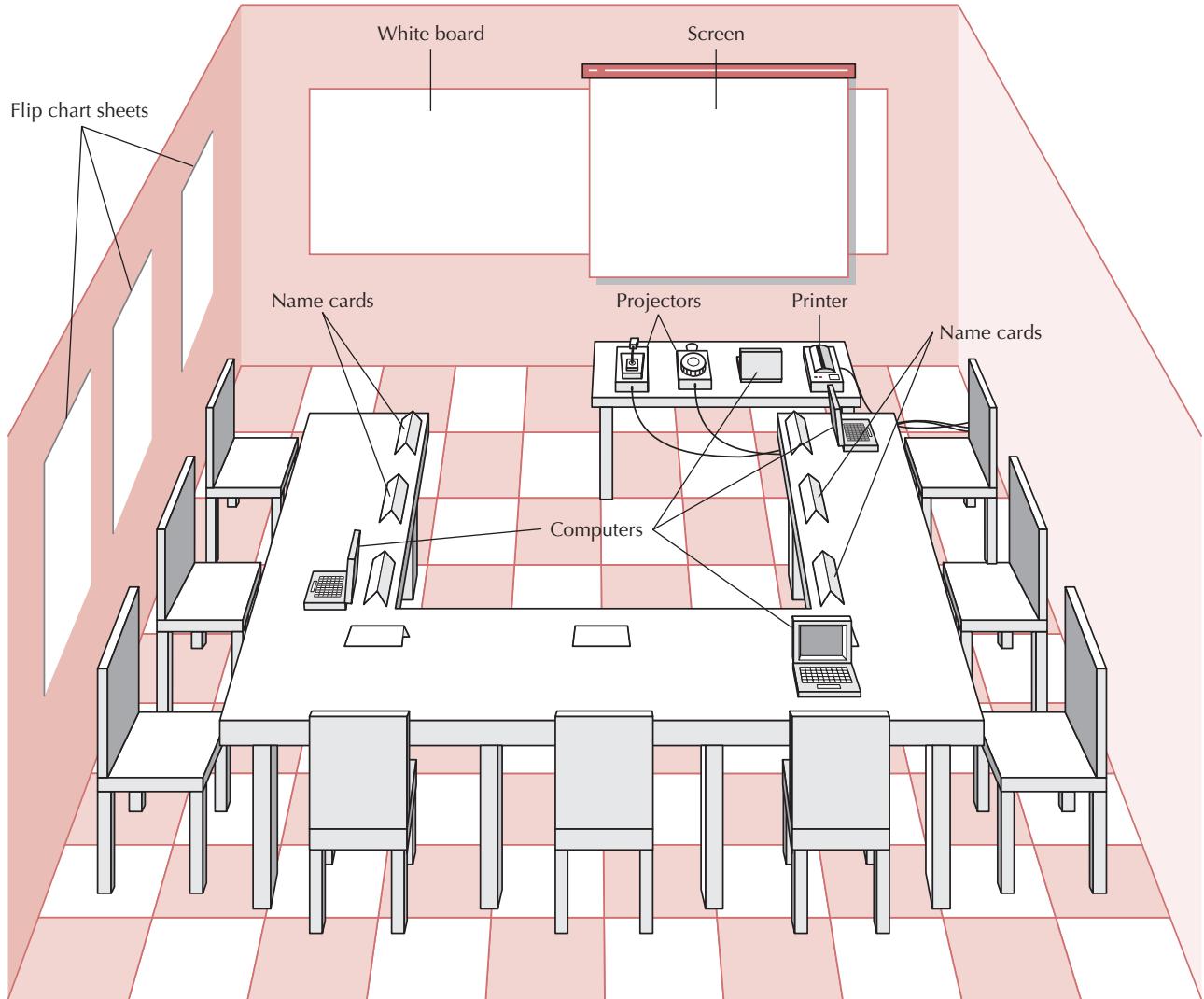


FIGURE 5-9 Joint Application Design Meeting Room

Ideally, the participants who are released from regular duties to attend the JAD sessions should be the very best people in that business unit. However, without strong management support, JAD sessions can fail because those selected to attend the JAD session are people who are less likely to be missed (i.e., the least competent people).

The facilitator should be someone who is an expert in JAD or e-JAD techniques and ideally someone who has experience with the business under discussion. In many cases, the JAD facilitator is a consultant external to the organization because the organization may not have a regular day-to-day need for JAD or e-JAD expertise. Developing and maintaining this expertise in-house can be expensive.

Designing the JAD Session JAD sessions can run from as little as a half day to several weeks, depending upon the size and scope of the project. In our experience, most JAD sessions tend to last five to ten days spread over a three-week period. Most e-JAD sessions tend to last one to four days in a one-week period. For example, the users and the analysts collectively can create analysis deliverables, such as the functional models, structural models, or the requirements definition.

As with interviewing, success depends upon a careful plan. JAD sessions usually are designed and structured using the same principles as interviews. Most JAD sessions are designed to collect specific information from users, and this requires the development of a set of questions prior to the meeting. A difference between JAD and interviewing is that all JAD sessions are structured—they *must* be carefully planned. In general, closed-ended questions are seldom used because they do not spark the open and frank discussion that is typical of JAD. In our experience, it is better to proceed top-down in JAD sessions when gathering information. Typically thirty minutes is allocated to each separate agenda item, and frequent breaks are scheduled throughout the day because participants tire easily.

Preparing for the JAD Session As with interviewing, it is important to prepare the analysts and participants for the JAD session. Because the sessions can go beyond the depth of a typical interview and are usually conducted off-site, participants can be more concerned about how to prepare. It is important that the participants understand what is expected of them. If the goal of the JAD session, for example, is to develop an understanding of the current system, then participants can bring procedure manuals and documents with them. If the goal is to identify improvements for a system, then they can think about how they would improve the system prior to the JAD Session.

Conducting the JAD Session Most JAD sessions try to follow a formal agenda, and most have formal *ground rules* that define appropriate behavior. Common ground rules include following the schedule, respecting others, opinions, accepting disagreement, and ensuring that only one person talks at a time.

The role of the JAD facilitator can be challenging. Many participants come to the JAD session with strong feelings about the system to be discussed. Channeling these feelings so that the session moves forward in a positive direction and getting participants to recognize and accept—but not necessarily agree on—opinions and situations different from their own requires significant expertise in systems analysis and design, JAD, and interpersonal skills. Few systems analysts attempt to facilitate JAD sessions without being trained in JAD techniques, and most apprentice with a skilled JAD facilitator before they attempt to lead their first session.

The JAD facilitator performs three key functions. First, he or she ensures that the group sticks to the agenda. The only reason to digress from the agenda is when it becomes clear to the facilitator, project leader, and project sponsor that the JAD session has produced some new information that is unexpected and requires the JAD session (and perhaps the project) to move in a new direction. When participants attempt to divert the discussion away from the agenda, the facilitator must be firm but polite in leading discussion back to the agenda and getting the group back on track.

Second, the facilitator must help the group understand the technical terms and jargon that surround the system development process, and help the participants understand the specific analysis techniques used. Participants are experts in their area, their part of the business, but they are not experts in systems analysis. The facilitator must therefore minimize the learning required and teach participants how to effectively provide the right information.

Third, the facilitator records the group's input on a public display area, which can be a whiteboard, flip chart, or computer display. He or she structures the information that the group provides and helps the group recognize key issues and important solutions. Under no circumstance should the facilitator insert his or her opinions into the discussion. The facilitator *must* remain neutral at all times and simply help the group through the process. The moment the facilitator offers an opinion on an issue, the group will no longer see him or her as a neutral party, but rather as someone who could be attempting to sway the group into some predetermined solution.

However, this does not mean that the facilitator should not try to help the group resolve issues. For example, if two items appear to be the same to the facilitator, the facilitator should not say, "I think these may be similar." Instead, the facilitator should ask, "Are these similar?" If the group decides they are, the facilitator can combine them and move on. However, if the group decides they are not similar (despite what the facilitator believes), the facilitator should accept the decision and move on. The group is *always* right, and the facilitator has no opinion.

Post JAD Follow-up As with interviews, a JAD postsession report is prepared and circulated among session attendees. The *postsession report* is essentially the same as the interview report in Figure 5-8. Since the JAD sessions are longer and provide more information, it usually takes a week or two after the JAD session before the report is complete.

Questionnaires

A *questionnaire* is a set of written questions for obtaining information from individuals. Questionnaires often are used when there is a large number of people from whom information and opinions are needed. In our experience, questionnaires are commonly used for systems intended for use outside of the organization (e.g., by customers or vendors) or for systems with business users spread across many geographic locations. Most people automatically think of paper when they think of questionnaires, but today more questionnaires are being distributed in electronic form, either via e-mail or on the Web. Electronic distribution can save a significant amount of money compared to distributing paper questionnaires.

Selecting Participants As with interviews and JAD sessions, the first step is to select the individuals to whom the questionnaire will be sent. However, it is not usual to select every person who could provide useful information. The standard approach is to select a *sample*, or subset, of people who are representative of the entire group. Sampling guidelines are discussed in most statistics books, and most business schools include courses that cover the topic, so we will not discuss it here. The important point in selecting a sample, however, is to realize that not everyone who receives a questionnaire will actually complete it. On average, only 30 to 50 percent of paper and e-mail questionnaires are returned. Response rates for Web-based questionnaires tend to be significantly lower (often only 5 to 30 percent).

YOUR TURN

5-5 JAD Practice

Organize yourselves into groups of four to seven people, and pick one person in each group to be the JAD facilitator. Using a blackboard, whiteboard, or flip chart, gather information about how the group performs some process

(e.g., working on a class assignment, making a sandwich, paying bills, getting to class). How did the JAD session go? Based on your experience, what are pros and cons of using JAD in a real organization?

**PRACTICAL****TIP****5-2 Managing Problems in JAD Sessions**

I have run more than a hundred JAD sessions and have learned several standard “facilitator tricks.” Here are some common problems and some ways to deal with them.

- **Reducing domination.** The facilitator should ensure that no one person dominates the group discussion. The only way to deal with someone who dominates is head on. During a break, approach the person, thank him or her for their insightful comments, and ask them to help you make sure that others also participate.
- **Encouraging noncontributors.** Drawing out people who have participated very little is challenging because you want to bring them into the conversation so that they will contribute again. The best approach is to ask a direct factual question that you are *certain* they can answer. And it helps to ask the question using some repetition to give them time to think. For example “Pat, I know you’ve worked shipping orders a long time. You’ve probably been in the Shipping Department longer than anyone else. Could you help us understand exactly what happens when an order is received in Shipping?”
- **Side discussions.** Sometimes participants engage in side conversations and fail to pay attention to the group. The easiest solution is simply to walk close to the people and continue to facilitate right in front of them. Few people will continue a side conversation when you are two feet from them and the entire group’s attention is on you and them.
- **Agenda merry-go-round.** The merry-go-round occurs when a group member keeps returning to the same issue every few minutes and won’t let go. One solution is to let the person have five minutes to ramble on about the issue while you carefully write down every point on a flip chart or computer file. This flip chart or file is then posted conspicuously on the wall. When the person brings up the issue again, you interrupt them, walk to the paper
- and ask them what to add. If they mention something already on the list, you quickly interrupt, point out that it is there, and ask what other information to add. Don’t let them repeat the same point, but write any new information.
- **Violent agreement.** Some of the worst disagreements occur when participants really agree on the issues but don’t realize that they agree because they are using different terms. An example is arguing whether a glass is half empty or half full; they agree on the facts, but can’t agree on the words. In this case, the facilitator has to translate the terms into different words and find common ground so the parties recognize that they really agree.
- **Unresolved conflict.** In some cases, participants don’t agree and can’t understand how to determine what alternatives are better. You can help by structuring the issue. Ask for criteria by which the group will identify a good alternative (e.g., “Suppose this idea really did improve customer service. How would I recognize the improved customer service?”). Then once you have a list of criteria, ask the group to assess the alternatives using them.
- **True conflict.** Sometimes, despite every attempt, participants just can’t agree on an issue. The solution is to postpone the discussion and move on. Document the issue as an “open issue” and list it prominently on a flip chart. Have the group return to the issue hours later. Often the issue will resolve itself by then and you haven’t wasted time on it. If the issue cannot be resolved later, move it to the list of issues to be decided by the project sponsor or some other more senior member of management.
- **Use humor.** Humor is one of the most power tools a facilitator has and thus must be used judiciously. The best JAD humor is always in context; never tell jokes but take the opportunity to find the humor in the situation.

—Alan Dennis

Designing the Questionnaire Developing good questions is critical for questionnaires because the information on a questionnaire cannot be immediately clarified for a confused respondent. Questions on questionnaires must be very clearly written and leave little room for misunderstanding, so closed-ended questions tend to be most commonly used. Questions must clearly enable the analyst to separate facts from opinions. Opinion questions often ask the respondent the extent to which they agree or disagree (e.g., “Are network problems common?”), while factual questions seek more precise values (e.g., “How often

does a network problem occur: once an hour, once a day, once a week?”). See Figure 5-10 for guidelines on questionnaire design.

Perhaps the most obvious issue—but one that is sometimes overlooked—is to have a clear understanding of how the information collected from the questionnaire will be analyzed and used. You must address this issue before you distribute the questionnaire, because it is too late afterward.

Questions should be relatively consistent in style, so that the respondent does not have to read instructions for each question before answering it. It is generally good practice to group related questions together to make them simpler to answer. Some experts suggest that questionnaires should start with questions important to respondents, so that the questionnaire immediately grabs their interest and induces them to answer it. Perhaps the most important step is to have several colleagues review the questionnaire and then pretest it with a few people drawn from the groups to whom it will be sent. It is surprising how often seemingly simple questions can be misunderstood.

Administering the questionnaire The key issue in administering the questionnaire is getting participants to complete the questionnaire and send it back. Dozens of marketing research books have been written about ways to improve response rates. Commonly used techniques include: clearly explaining why the questionnaire is being conducted and why the respondent has been selected; stating a date by which the questionnaire is to be returned; offering an inducement to complete the questionnaire (e.g., a free pen); and offering to supply a summary

YOUR TURN

5-6 Questionnaire Practice

Organize yourselves into small groups. Have each person develop a short questionnaire to collect information about the frequency in which group members perform some process (e.g., working on a class assignment, making a sandwich, paying bills, getting to class), how long it takes them, how they feel about the process, and opportunities for improving the process.

Once everyone has completed his or her questionnaire, ask each member to pass it to the right, and then complete his or her neighbor’s questionnaire. Pass the questionnaire back to the creator when it is completed.

Questions:

1. How did the questionnaire you completed differ from the one you created?
2. What are the strengths of each questionnaire?
3. How would you analyze the survey results if you had received fifty responses?
4. What would you change about the questionnaire that you developed?

- Begin with nonthreatening and interesting questions.
- Group items into logically coherent sections.
- Do not put important items at the very end of the questionnaire.
- Do not crowd a page with too many items.
- Avoid abbreviations.
- Avoid biased or suggestive items or terms.
- Number questions to avoid confusion.
- Pretest the questionnaire to identify confusing questions.
- Provide anonymity to respondents.

FIGURE 5-10
Good Questionnaire Design

of the questionnaire responses. Systems analysts have additional techniques to improve response rates inside the organization, such as personally handing out the questionnaire and personally contacting those who have not returned them after a week or two, as well as requesting the respondents' supervisors to administer the questionnaires in a group meeting.

Questionnaire Follow-up It is helpful to process the returned questionnaires and develop a questionnaire report soon after the questionnaire deadline. This ensures that the analysis process proceeds in a timely fashion and that respondents who requested copies of the results receive them promptly.

Document Analysis

Project teams often use *document analysis* to understand the as-is system. Under ideal circumstances, the project team that developed the existing system will have produced documentation, which was then updated by all subsequent projects. In this case, the project team can start by reviewing the documentation and examining the system itself.

Unfortunately, most systems are not well documented because project teams fail to document their projects along the way, and when the projects are over—there is no time to go back and document. Therefore, there may not be much technical documentation about the current systems available, or it may not contain updated information about recent system changes. However, there are many helpful documents that do exist in the organization: paper reports, memorandums, policy manuals, user training manuals, organization charts, and forms.

But these documents (forms, reports, policy manuals, organization charts) only tell part of the story. They represent the *formal system* that the organization uses. Quite often, the “real” or *informal system* differs from the formal one, and these differences, particularly large ones, give strong indications of what needs to be changed. For example, forms or reports that are never used likely should be eliminated. Likewise, boxes or questions on forms that are never filled in (or are used for other purposes) should be rethought. See Figure 5-11 for an example of how a document can be interpreted.

The most powerful indication that the system needs to be changed is when users create their own forms or add additional information to existing ones. Such changes clearly demonstrate the need for improvements to existing systems. Thus, it is useful to review both blank and completed forms to identify these deviations. Likewise, when users access multiple reports to satisfy their information needs, it is a clear sign that new information or new information formats are needed.

Observation

Observation, the act of watching processes being performed, is a powerful tool for gathering information about the as-is system because it enables the analyst to see the reality of a situation, rather than listening to others describe it in interviews or JAD sessions. Several research studies have shown that many managers really do not remember how they work and how they allocate their time. (Quick, how many hours did you spend last week on each of your courses?) Observation is a good way to check the validity of information gathered from indirect sources such as interviews and questionnaires.⁹

In many ways, the analyst becomes an anthropologist as he or she walks through the organization and observes the business system as it functions. The goal is to keep a low profile, to not interrupt those working, and to not influence those being observed. Nonetheless, it is important to understand that what analysts observe may not be the normal

⁹ A good book on observation is James P. Spradley, *Participant Observation* (New York, NY: Holt, Rinehart, and Winston, 1980).

**CENTRAL VETERINARY CLINIC
Patient Information Card**

Name:	<u>Duffy</u>	Pat Smith
Pet's Name:	Buffy	Collie 7/6/99
Address:	100 Central Court. Apartment 10	
Toronto, Ontario K7L 3N6		
Phone Number:	416-	555-3400
Do you have insurance:	yes	
Insurance Company:	Pet's Mutual	
Policy Number:	KA-5493243	

TEMPLATE
can be found at
www.wiley.com/college/dennis

The customer made a mistake. This should be labeled **Owner's Name** to prevent confusion.

The staff had to add additional information about the type of animal and the animal's date of birth. This information should be added to the new form in the to-be system.

The customer did not include area code in the phone number. This should be made more clear.

FIGURE 5-11
Performing a Document Analysis

day-to-day routine because people tend to be extremely careful in their behavior when they are being watched. Even though normal practice may be to break formal organizational rules, the observer is unlikely to see this. (Remember how you drove the last time a police car followed you?) Thus, what you see may *not* be what you get.

Observation is often used to supplement interview information. The location of a person's office and its furnishings gives clues as to their power and influence in the organization, and can be used to support or refute information given in an interview. For example, an analyst might become skeptical of someone who claims to use the existing computer system extensively if the computer is never turned on while the analyst visits. In most cases, observation will support the information that users provide in interviews. When it does not, it is an important signal that extra care must be taken in analyzing the business system.

Selecting the Appropriate Techniques

Each of the requirements-gathering techniques just discussed has strengths and weaknesses. No one technique is always better than the others, and in practice most projects use a combination of techniques. Thus, it is important to understand the strengths and weaknesses of each technique and when to use each (see Figure 5-12). One issue not discussed is that of the analysts' experience. In general, document analysis and observation require the least amount of training, while JAD sessions are the most challenging.

	Interviews	Joint Application Design	Questionnaires	Document Analysis	Observation
Type of information	As-is, improvements, to-be	As-is, improvements, to-be	As-is, improvements	As-is	As-is
Depth of information	High	High	Medium	Low	Low
Breadth of information	Low	Medium	High	High	Low
Integration of information	Low	High	Low	Low	Low
User involvement	Medium	High	Low	Low	Low
Cost	Medium	Low–Medium	Low	Low	Low–Medium

FIGURE 5-12 Table of Requirements-Gathering Techniques

CONCEPTS

5-E Publix Credit Card Forms

IN ACTION

At my neighborhood Publix grocery store, the cashiers always hand write the total amount of the charge on every credit card charge form, even though it is printed on the form. Why? Because the "back office" staff people who reconcile the cash in the cash drawers with the amount sold at the end of each shift find it hard to read the small print on the credit card forms. Writing in large print makes it easier for them to add the values up. However, cashiers sometimes make

mistakes and write the wrong amount on the forms, which causes problems.

—Barbara Wixom

Questions:

1. What does the credit card charge form indicate about the existing system?
2. How can you make improvements with a new system?

YOUR TURN

5-7 Observation Practice

Visit the library at your college or university and observe how the book check-out process occurs. First watch several students checking books out, and then check one out yourself. Prepare a brief summary report of your observations.

When you return to class, share your observations with others. You may notice that not all the reports present the same information. Why? How would the information be different had you used the interview or JAD technique?

Type of Information The first characteristic is type of information. Some techniques are more suited for use at different stages of the analysis process, whether understanding the as-is system, identifying improvements, or developing the to-be system. Interviews and JAD are commonly used in all three stages. In contrast, document analysis and observation usually are most helpful for understanding the as-is, although occasionally they provide information about current problems that need to be improved. Questionnaires are often used to gather information about the as-is system, as well as general information about improvements.

Depth of information The depth of information refers to how rich and detailed the information is that the technique usually produces, and the extent to which the technique is useful at obtaining not only facts and opinions, but also an understanding of *why* those facts and opinions exist. Interviews and JAD sessions are very useful at providing a good depth of rich and detailed information and helping the analyst to understand the reasons behind them. At the other extreme, document analysis and observation are useful for obtaining facts, but little beyond that. Questionnaires can provide a medium depth of information, soliciting both facts and opinions with little understanding of why.

Breadth of Information Breadth of information refers to the range of information and information sources that can be easily collected using that technique. Questionnaires and document analysis both are easily capable of soliciting a wide range of information from a large number of information sources. In contrast, interviews and observation require the analyst to visit each information source individually and, therefore, take more time. JAD sessions are in the middle because many information sources are brought together at the same time.

Integration of Information One of the most challenging aspects of requirements gathering is the integration of information from different sources. Simply put, different people can provide conflicting information. Combining this information and attempting to resolve differences in opinions or facts is usually very time consuming because it means contacting each information source in turn, explaining the discrepancy, and attempting to refine the information. In many cases, the individual wrongly perceives that the analyst is challenging his or her information, when in fact it is another user in the organization. This can make the user defensive and make it hard to resolve the differences.

All techniques suffer integration problems to some degree, but JAD sessions are designed to improve integration because all information is integrated when it is collected, not afterward. If two users provide conflicting information, the conflict becomes immediately obvious, as does the source of the conflict. The immediate integration of information is the single most important benefit of JAD that distinguishes it from other techniques, and this is why most organizations use JAD for important projects.

User Involvement User involvement refers to the amount of time and energy the intended users of the new system must devote to the analysis process. It is generally agreed that as users become more involved in the analysis process, the greater the chance of success. However, user involvement can have a significant cost, and not all users are willing to contribute valuable time and energy. Questionnaires, document analysis, and observation place the least burden on users, while JAD sessions require the greatest effort.

Cost Cost is always an important consideration. In general, questionnaires, document analysis, and observation are low cost techniques (although observation can be quite time consuming). The low cost does not imply that they are more or less effective than the other

techniques. We regard interviews and JAD sessions as having moderate costs. In general, JAD sessions are much more expensive initially, because they require many users to be absent from their offices for significant periods of time, and they often involve highly paid consultants. However, JAD sessions significantly reduce the time spent in information integration and thus cost less in the long term.

Combining Techniques In practice, requirements gathering combines a series of different techniques. Most analysts start by using interviews with senior manager(s) to gain an understanding of the project and the “big picture” issues. From this, the scope becomes clear as to whether large or small changes are anticipated. These are often followed with analysis of documents and policies to gain some understanding of the as-is system. Usually interviews come next to gather the rest of the information needed for the as-is system.

In our experience, identifying improvements is most commonly done using JAD sessions because the JAD session enables the users and key stakeholders to work together through an analysis technique and come to a shared understanding of the possibilities for the to-be system. Occasionally, these JAD sessions are followed by questionnaires sent to a much wider set of users or potential users to see whether the opinions of those who participated in the JAD sessions are widely shared.

Developing the concept for the to-be system is often done through interviews with senior managers, followed by JAD sessions with users of all levels to make sure the key needs of the new system are well understood.

APPLYING THE CONCEPTS AT CD SELECTIONS

Once the CD Selections approval committee approved the system proposal and feasibility analysis, the project team began performing analysis activities. These included gathering requirements using a variety of techniques, and analyzing the requirements that were gathered. An Internet marketing and sales consultant, Chris Campbell, was hired to advise Alec, Margaret, and the project team during the analysis phase. Some highlights of the project team’s activities are presented next.

Requirements Analysis Techniques

Margaret suggested that the project team conduct several JAD sessions with store managers, marketing analysts, and Web-savvy members of the IT staff. Together, the groups could work through some BPI techniques and brainstorm how improvements could be made to the current order process using a new Web-based system.

Alec facilitated three JAD sessions that were conducted over the course of a week. Alec’s past facilitation experience helped the eight-person meetings run smoothly and stay on track. First, Alec used technology analysis and suggested several important Web technologies that could be used for the system. The JAD session generated ideas about how CD Selections could apply each of the technologies to the Internet sales system project. Alec had the group categorize the ideas into three sets: “definite” ideas that would have a good probability of providing business value; “possible” ideas that might add business value; and “unlikely” ideas.

Next, Alec applied informal benchmarking by introducing the Web sites of several leading retailers and pointing out the features that they offered online. He selected some sites based on their success with Internet sales, and others based on their similarity to the vision for CD Selections’ new system. The group discussed the features that were common across most retailers versus unique functionality, and they created a list of suggested business requirements for the project team.

Requirements-Gathering Techniques

Alec believed that it would be important to understand the order processes and systems that already existed in the organization because they would have to be closely integrated with the Internet sales system. Three requirements-gathering techniques proved to be helpful in understanding the current systems and processes—document analysis, interviews, and observation.

First, the project team collected existing reports (e.g., order forms, screenshots of the online order screens) and system documentation that shed light on the as-is system. They were able to gather a good amount of information about the brick-and-mortar order processes and systems in this way. When questions arose, they conducted short interviews with the person who provided the documentation for clarification.

Next, Alec interviewed the senior analysts for the order and inventory systems to get a better understanding of how those systems worked. He asked if they had any ideas for the new system, as well as any integration issues that would need to be addressed. Alex also interviewed a contact from the ISP and the IT person who supported CD Selections' current Web site—both provided information about the existing communications infrastructure at CD Selections and its Web capabilities. Finally, Alex spent a half day visiting two of the retail stores and observing exactly how the order and hold processes worked in the brick-and-mortar facilities.

Requirements Definition

Throughout all of these activities, the project team collected information and tried to identify the business requirements for the system from the information. As the project progressed, requirements were added to the requirements definition and grouped by requirement type. When questions arose, they worked with Margaret, Chris, and Alec to confirm that requirements were in scope. The requirements that fell outside of the scope of the current system were typed into a separate document that would be saved for future use.

At the end of analysis, the requirements definition was distributed to Margaret, two marketing employees who would work with the system on the business side, and several retail store managers. This group then met for a two-day JAD session to clarify, finalize, and prioritize business requirements and to create use cases (Chapter 6) to show how the system would be used.

The project team also spent time creating structural and behavioral models (Chapters 7 and 8) that depicted the objects in the future system. Members of marketing and IT departments reviewed the documents during interviews with the project team. Figure 5-13 shows a portion of the final requirements definition.

System Proposal

Alec reviewed the requirements definition and the other deliverables that the project team created during the analysis phase. Given Margaret's desire to have the system operating before next year's Christmas season, Alec decided to timebox the project, and he determined what functionality could be included in the system by that schedule deadline (see Chapter 3). He suggested that the project team develop the system in three versions rather than attempting to develop a complete system that provided all the features initially (Phased Development, see Chapter 2). The first version, to be operational well before the holidays, would implement a “basic” system that would have the “standard” order features of other Internet retailers. The second version, planned for late spring or early summer, would have several features unique to CD Selections. The third version would add more “advanced” features, such as the ability to listen to a sample of music over the Internet, to find similar CDs, and to write reviews.

Nonfunctional Requirements

1. Operational Requirements

- 1.1 The Internet sales system will draw information from the main CD information database, which contains basic information about CDs (e.g., title, artist, ID number, price, quantity in inventory). The Internet sales system will not write information to the main CD information database.
- 1.2 The Internet sales system will store orders for new CDs in the special order system and will rely on the special order system to complete the special orders generated.
- 1.3 A new module for the in-store system will be written to manage the “holds” generated by the Internet sales system. The requirements for this new module will be documented as part of the Internet sales system because they are necessary for the Internet sales system to function.

2. Performance Requirements

No special requirements performance requirements are anticipated

3. Security Requirements

No special security requirements are anticipated

4. Cultural and Political Requirements

No special cultural and political requirements are anticipated

Functional Requirements

1. Maintain CD Information

- 1.1 The Internet sales system will need a database of basic information about the CDs that it can sell over the Internet, similar to the CD database at each of the retail stores (e.g., title, artist, id number, price, quantity in inventory).
- 1.2 Every day, the Internet sales system will receive an update from the distribution system that that will be used to update this CD database. Some new CDs will be added, some will be deleted, and others will be revised (e.g., a new price).
- 1.3 The electronic marketing (EM) manager (a position that will need to be created) will also have the ability to update information (e.g., prices for sales).

FIGURE 5-13 CD Selections Requirements Definition (Continues)

Alec revised the workplan accordingly, and he worked with Margaret and the folks in Marketing to review the feasibility analysis and update it where appropriate. All of the deliverables from the project were then combined into a system proposal and submitted to the approval committee. Figure 5-14 shows the outline of the CD Selections system proposal. Margaret and Alec met with the committee and presented the highlights of what was learned during the analysis phase and the final concept of the new system. Based on the proposal and presentation, the approval committee decided that they would continue to fund the Internet sales system.

SUMMARY

Analysis

Analysis focuses on capturing the business requirements for the system. It identifies the “what” of the system and leads directly into design, during which the “how” of the system is determined. Many deliverables are created during analysis, including the requirements definition, functional models, structural models, and behavioral models. At the end of analysis, all of these deliverables, along with revised planning and project management deliverables, are combined into a system proposal and submitted to the approval committee for a decision regarding whether to move ahead with the project.

2. Maintain CD Marketing Information

- 2.1 The Internet sales system provides an additional opportunity to market CDs to current and new customers. The system will provide a database of marketing materials about selected CDs that will help Web users learn more about them (e.g., music reviews, links to Web sites, artist information, and sample sound clips). When information about a CD that has additional marketing information is displayed, a link will be provided to the additional information.
- 2.2 Marketing materials will be supplied primarily by vendors and record labels so that we can better promote their CDs. The EM manager of the marketing department will determine what marketing materials will be placed in the system and will be responsible for adding, changing, and deleting the materials.

3. Place CD Orders

- 3.1 Customers will access the Internet sales system to look for CDs of interest. Some customers will search for specific CDs or CDs by specific artists, while other customers will want to browse for interesting CDs in certain categories (e.g., rock, jazz, classical).
- 3.2 When the customer has found all the CDs he or she wants, the customer will "check out" by providing personal information (e.g., name, e-mail, address, credit card), and information regarding the order (e.g., the CDs to purchase, and the quantity for each item).
- 3.3 The system will verify the customer's credit card information with an online credit card clearance center and either accept the order or reject it.
- 3.4 Customers will also be able check to see if their preferred stores have the CDs in stock. They will use zip code to find stores close to their location. If the CD is available at a preferred store, a customer can immediately place a hold on the CD in stock and then come into the store and pick it up.
- 3.5 If the CD is not available in the customer's preferred store, the customer can request that the CD be special ordered to that store for later pickup. The customer will be notified by e-mail when the requested CD arrives at the requested store; the CD will be placed on hold (which will again expire after 7 days). This process will work similarly to the current special order systems already available in the regular stores.
- 3.6 Alternatively, the customer can mail order the CD (see requirement 4).

4. Fill Mail Orders

- 4.1 When a CD is mail ordered, the Internet sales system will send the mail order to the mail order distribution system.
- 4.2 The mail order distribution system will handle the actual sending of CDs to customers; it will notify the Internet sales system and e-mail the customer.
- 4.3 Weekly reports can be run by the EM manager to check the order status.

FIGURE 5-13 CD Selections Requirements Definition (Continued)

Requirements Determination

Requirements determination is the part of analysis whereby the project team turns the very high-level explanation of the business requirements stated in the system request into a more precise list of requirements. A requirement is simply a statement of what the system must do or what characteristic it needs to have. Business requirements describe the "what" of the systems, and system requirements describe "how" the system will be implemented. A functional requirement relates directly to a process the system has to perform or information it needs to contain. Nonfunctional requirements refer to behavioral properties that the system must have, such as performance and usability. All of the functional and non-functional business requirements that fit within the scope of the system are written in the requirements definition, which is used to create other analysis deliverables and leads to the initial design for the new system.

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 5-14
Outline of the CD Selections System Proposal

1. Table of Contents
2. Executive Summary
A summary of all the essential information in the proposal so a busy executive can read it quickly and decide what parts of the plan to read in more depth.
3. System Request
The revised system request form (see Chapter 3).
4. Workplan
The original workplan, revised after having completed the analysis phase (see Chapter 4)
5. Requirements Definition
A list of the functional and nonfunctional business requirements for the system (this chapter).
6. Feasibility Analysis
A revised feasibility analysis, using the information from the analysis phase (see Chapter 3).
7. Functional Models
An activity diagram and a set of use cases that illustrate the basic processes that the system needs to support (see Chapter 6).
8. Structural Models
A set of structural models for the to-be system (see Chapter 7). This may include structural models of the current as-is system that will be replaced.
9. Behavioral Models
A set of behavioral models for the to-be system (see Chapter 8). This may include behavioral models of the as-is system that will be replaced.
Appendices
These contain additional material relevant to the proposal, often used to support the recommended system. This might include results of a questionnaire survey or interviews, industry reports and statistics, and so on.

Requirements Analysis Techniques

The basic process of analysis is divided into three steps: understanding the as-is system, identifying improvements, and developing requirements for the to-be system. Three requirements analysis techniques—business process automation, business process improvement, or business process reengineering—help the analyst lead users through the three (or two) analysis steps so that the vision of the system can be developed. Business process automation (BPA) means leaving the basic way in which the organization operates unchanged, and using computer technology to do some of the work. Problem analysis and root cause analysis are two popular BPA techniques. Business process improvement (BPI) means making moderate changes to the way in which the organization operates to take advantage of new opportunities offered by technology or to copy what competitors are doing. Duration analysis, activity-based costing, and information benchmarking are three popular BPI activities. Business process reengineering (BPR) means changing the fundamental way in which the organization operates. Outcome analysis, technology analysis, and activity elimination are three popular BPR activities.

Requirements-Gathering Techniques

Five techniques can be used to gather the business requirements for the proposed system: interviews, joint application development, questionnaires, document analysis, and observation. Interviews involve meeting one or more people and asking them questions. There are five basic steps to the interview process: selecting interviewees, designing interview

questions, preparing for the interview, conducting the interview, and postinterview follow-up. Joint application development (JAD) allows the project team, users, and management to work together to identify requirements for the system. Electronic JAD attempts to overcome common problems associated with groups by using groupware. A questionnaire is a set of written questions for obtaining information from individuals. Questionnaires often are used when there is a large number of people from whom information and opinions are needed. Document analysis entails reviewing the documentation and examining the system itself. It can provide insights into the formal and informal system. Observation, the act of watching processes being performed, is a powerful tool for gathering information about the as-is system because it enables the analyst to see the reality of a situation firsthand.

KEY TERMS

Activity elimination	Informal benchmarking	Questionnaire
Activity-based costing	Informal system	Requirement
Analysis	Interpersonal skill	Requirements definition
As-is system	Interview	Requirements determination
Benchmarking	Interview notes	Risk
Bottom-up interview	Interview report	Root cause
Breadth of analysis	Interview schedule	Root cause analysis
Business process automation (BPA)	Joint application development (JAD)	Sample
Business process improvement (BPI)	Nonfunctional requirements	Scribe
Business process reengineering (BPR)	Observation	Stakeholders
Business requirement	Open-ended question	Structured interview
Closed-ended question	Outcome analysis	Symptom
Critical thinking skills	Parallelization	System proposal
Document analysis	Problem analysis	System requirements
Duration analysis	Process integration	Technology analysis
Electronic JAD (e-JAD)	Postsession report	To-be system
Facilitator	Potential business value	Top-down interview
Formal system	Probing question	Unstructured interview
Functional requirement	Problem analysis	Walkthrough
Ground rule	Project cost	

QUESTIONS

- What are the key deliverables that are created during the analysis phase? What is the final deliverable from the analysis phase, and what does it contain?
- Explain the difference between an as-is system and a to-be system.
- What is the purpose of the requirements definition?
- What are the three basic steps of the analysis process? Which step is sometimes skipped or done in a cursory fashion? Why?
- Compare and contrast the business goals of BPA, BPI, and BPR.
- Compare and contrast problem analysis and root cause analysis. Under what conditions would you use problem analysis? Under what conditions would you use root cause analysis?
- Compare and contrast duration analysis and activity-based costing.
- Assuming time and money were not important concerns, would BPR projects benefit from additional time spent understanding the as-is system? Why or why not?
- What are the important factors in selecting an appropriate analysis strategy?
- Describe the five major steps in conducting interviews.
- Explain the difference between a closed-ended question, an open-ended question, and a probing question. When would you use each?

12. Explain the differences between unstructured interviews and structured interviews. When would you use each approach?
13. Explain the difference between a top-down and bottom-up interview approach. When would you use each approach?
14. How are participants selected for interviews and JAD sessions?
15. How can you differentiate between facts and opinions? Why can both be useful?
16. Describe the five major steps in conducting JAD sessions.
17. How does a JAD facilitator differ from a scribe?
18. What are the three primary things that a facilitator does in conducting the JAD session?
19. What is e-JAD, and why might a company be interested in using it?
20. How does designing questions for questionnaires differ from designing questions for interviews or JAD sessions?
21. What are typical response rates for questionnaires and how can you improve them?
22. Describe document analysis.
23. How does the formal system differ from the informal system? How does document analysis help you understand both?
24. What are the key aspects of using observation in the information-gathering process?
25. Explain factors that can be used to select information-gathering techniques

EXERCISES

- A. Review the Amazon.com Web site. Develop the requirements definition for the site. Create a list of functional business requirements that the system meets. What different kinds of nonfunctional business requirements does the system meet? Provide examples for each kind.
- B. Pretend that you are going to build a new system that automates or improves the interview process for the Career Services Department of your school. Develop a requirements definition for the new system. Include both functional and nonfunctional system requirements. Pretend you will release the system in three different versions. Prioritize the requirements accordingly.
- C. Describe in very general terms the as-is business process for registering for classes at your university. What BPA technique would you use to identify improvements? With whom would you use the BPA technique? What requirements-gathering technique would help you apply the BPA technique? List some example improvements that would you expect to find.
- D. Describe in very general terms the as-is business process for registering for classes at your university. What BPI technique would you use to identify improvements? With whom would you use the BPI technique? What requirements-gathering technique would help you apply the BPI technique? List some example improvements that you would expect to find.
- E. Describe in very general terms the as-is business process for registering for classes at your university. What BPR technique would you use to identify improvements? With whom would you use the BPR technique? What requirements-gathering technique
- would help you apply the BPR technique? List some example improvements that would you expect to find.
- F. Suppose your university is having a dramatic increase in enrollment and is having difficulty finding enough seats in courses for students so they can take courses required for graduation. Perform a technology analysis to identify new ways to help students complete their studies and graduate.
- G. Suppose you are the analyst charged with developing a new system for the university bookstore with which students can order books online and have them delivered to their dorms and off-campus housing. What requirements-gathering techniques will you use? Describe in detail how you would apply the techniques.
- H. Suppose you are the analyst charged with developing a new system to help senior managers make better strategic decisions. What requirements-gathering techniques will you use? Describe in detail how you would apply the techniques.
- I. Find a partner and interview each other about what tasks you/they did in the last job held (full-time, part-time, past or current). If you haven't worked before, then assume your job is being a student. Before you do this, develop a brief interview plan. After your partner interviews you, identify the type of interview, interview approach, and types of questions used.
- J. Find a group of students and run a sixty-minute JAD session on improving alumni relations at your university. Develop a brief JAD plan, select two techniques that will help identify improvements, and then develop an agenda. Conduct the session using the agenda, and write your postsession report.

- K. Find a questionnaire on the Web that has been created to capture customer information. Describe the purpose of the survey, the way questions are worded, and how the questions have been organized. How can it be improved? How will the responses be analyzed?
- L. Develop a questionnaire that will help gather information regarding processes at a popular restaurant, or the college cafeteria (e.g., ordering, customer service).

Give the questionnaire to ten to fifteen students, analyze the responses, and write a brief report that describes the results.

- M. Contact the Career Services Department at your university and find all the pertinent documents designed to help students find permanent and/or part-time jobs. Analyze the documents and write a brief report.

MINICASES

- The State Firefighter's Association has a membership of 15,000. The purpose of the organization is to provide some financial support to the families of deceased member firefighters and to organize a conference each year bringing together firefighters from all over the state. Annually members are billed dues and calls. "Calls" are additional funds required to take care of payments made to the families of deceased members. The bookkeeping work for the association is handled by the elected treasurer, Bob Smith, although it is widely known that his wife, Laura, does all of the work. Bob runs unopposed each year at the election, since no one wants to take over the tedious and time-consuming job of tracking memberships. Bob is paid a stipend of \$8,000 per year, but his wife spends well over twenty hours per week on the job. The organization however, is not happy with their performance.

A computer system is used to track the billing and receipt of funds. This system was developed in 1984 by a Computer Science student and his father. The system is a DOS-based system written using dBase 3. The most immediate problem facing the treasurer and his wife is the fact that the software package no longer exists, and there is no one around who knows how to maintain the system. One query in particular takes seventeen hours to run. Over the years, they have just avoided running this query, although the information in it would be quite useful. Questions from members concerning their statements cannot be easily answered. Usually Bob or Laura just jot down the inquiry and return a call with the answer. Sometimes it takes three to five hours to find the information needed to answer the question. Often, they have to perform calculations manually since the system was not programmed to handle certain types of queries. When member information is entered into the system, each field is presented one at a time. This makes it very difficult to return to a field and correct a value that was entered. Sometimes a new

member is entered but disappears from the records. The report of membership used in the conference materials does not alphabetize members by city. Only cities are listed in the correct order.

What requirements analysis strategy or strategies would you recommend for this situation? Explain your answer.

- Brian Callahan, IS Project Manager, is just about ready to depart for an urgent meeting called by Joe Campbell, Manager of Manufacturing Operations. A major BPI project sponsored by Joe recently cleared the approval hurdle, and Brian helped bring the project through project initiation. Now that the approval committee has given the go-ahead, Brian has been working on the project's analysis plan.

One evening, while playing golf with a friend who works in the Manufacturing Operations Department, Brian learned that Joe wants to push the project's time frame up from Brian's original estimate of thirteen months. Brian's friend overheard Joe say, "I can't see why that IS project team needs to spend all that time 'analyzing' things. They've got two weeks scheduled just to look at the existing system! That seems like a real waste. I want that team to get going on building my system."

Because Brian has a little inside knowledge about Joe's agenda for this meeting, he has been considering how to handle Joe. What do you suggest Brian tell Joe?

- Barry has recently been assigned to a project team that will be developing a new retail store management system for a chain of submarine sandwich shops. Barry has several years of experience in programming but has not done much analysis in his career. He was a little nervous about the new work he would be doing, but was confident he could handle any assignment he was given.

One of Barry's first assignments was to visit one of the submarine sandwich shops and prepare an observation report on how the store operates. Barry

planned to arrive at the store around noon, but he chose a store in an area of town he was unfamiliar with, and due to traffic delays and difficulty in finding the store, he did not arrive until 1:30 PM. The store manager was not expecting him and refused to let a stranger behind the counter until Barry had him contact the project sponsor (the Director of Store Management) back at company headquarters to verify who he was and what his purpose was.

After finally securing permission to observe, Barry stationed himself prominently in the work area behind the counter so that he could see everything. The staff had to maneuver around him as they went about their tasks, and there were only minor occasional collisions. Barry noticed that the store staff seemed to be going about their work very slowly and deliberately, but he supposed that was because the store wasn't very busy. At first, Barry questioned each worker about what he or she was doing, but the store manager eventually asked him not to interrupt their work so much—he was interfering with their service to the customers.

By 3:30 PM, Barry was a little bored. He decided to leave, figuring he could get back to the office and prepare his report before 5:00 PM that day. He was sure his team leader would be pleased with his quick completion of his assignment. As he drove, he reflected, "There really won't be much to say in this report. All they do is take the order, make the sandwich, collect the payment, and hand over the order. It's really simple!" Barry's confidence in his analytical skills soared as he anticipated his team leader's praise.

Back at the store, the store manager shook his head, commenting to his staff, "He comes here at the slowest time of day on the slowest day of the week. He never even looked at all the work I was doing in the back room while he was here—summarizing yesterday's sales,

checking inventory on hand, making up resupply orders for the weekend ... plus he never even considered our store opening and closing procedures. I hate to think that the new store management system is going to be built by someone like that. I'd better contact Chuck (the Director of Store Management) and let him know what went on here today." Evaluate Barry's conduct of the observation assignment.

4. Anne has been given the task of conducting a survey of sales clerks who will be using a new order entry system being developed for a household products catalog company. The goal of the survey is to identify the clerks' opinions on the strengths and weaknesses of the current system. There are about fifty clerks who work in three different cities, so a survey seemed like an ideal way of gathering the needed information from the clerks.

Anne developed the questionnaire carefully and pretested it on several sales supervisors who were available at corporate headquarters. After revising it based on their suggestions, she sent a paper version of the questionnaire to each clerk, asking that it be returned within one week. After one week, she had only three completed questionnaires returned. After another week, Anne received just two more completed questionnaires. Feeling somewhat desperate, Anne then sent out an e-mail version of the questionnaire, again to all the clerks, asking them to respond to the questionnaire by e-mail as soon as possible. She received two e-mail questionnaires and three messages from clerks who had completed the paper version expressing annoyance at being bothered with the same questionnaire a second time. At this point, Anne has just a 14 percent response rate, which she is sure will not please her team leader. What suggestions do you have that could have improved Anne's response rate to the questionnaire?

CHAPTER 6

FUNCTIONAL MODELING

Functional models describe business processes and the interaction of an information system with its environment. In object-oriented systems development, two types of models are used to describe the functionality of an information system: activity diagrams and use cases. Activity diagrams support the logical modeling of business processes and workflows. Use cases are used to describe the basic functions of the information system. Both activity diagrams and use cases can be used to describe the current as-is system and the to-be system being developed. This chapter describes functional modeling as a means to document and understand requirements, and to understand the functional or external behavior of the system. This chapter also introduces use case points as a basis for project size estimation.

OBJECTIVES:

- Understand the rules and style guidelines for activity diagrams.
- Understand the rules and style guidelines for use cases and use case diagrams.
- Understand the process used to create use cases and use case diagrams.
- Be able to create functional models using activity diagrams, use cases, and use case diagrams.
- Become familiar with the use of use case points.

CHAPTER OUTLINE

Introduction	Identify the Major Use Cases
Business Process Modeling with Activity Diagrams	Expand the Major Use Cases
Elements of an Activity Diagram	Confirm the Major Use Cases
Guidelines for Creating Activity Diagrams	Create the Use Case Diagram
Use Case Descriptions	Refining Project Size and Effort
Types of Use Cases	Estimation using Use Case Points
Elements of a Use Case Description	Applying the Concepts at CD Selections
Guidelines for Creating Use Case Descriptions	Business Process Modeling with Activity Diagrams
Use Case Diagrams	Identifying the Major Use Cases
Actor	Expanding the Major Use Cases
Association	Confirming the Major Use Cases
Use Case	Creating the Use Case Diagram
Subject Boundary	Refining the Project Size and Effort
Creating Use Case Descriptions and Use Case Diagrams	Estimation Using Use Case Points
	Summary

INTRODUCTION

The previous chapter discussed the more popular requirements-gathering techniques such as interviewing, JAD, and observation. Using these techniques, the analyst determined the requirements and created a requirements definition. The requirements definition defined what the system is to do. In this chapter, we discuss how the information that is gathered using these techniques is organized and presented in the form of activity diagrams and use cases. Since UML has been accepted as the standard notation by the Object Management Group (OMG), almost all object-oriented development projects today utilize activity diagrams and use cases to document and organize the requirements that are obtained during the analysis phase.¹

An *activity diagram* can be used for any type of process modeling activity.² In this chapter, we describe their use in the context of business process modeling. *Process models* depict how a business system operates. They illustrate the processes or activities that are performed and how objects (data) move among them. A process model can be used to document the current system (i.e., as-is system) or the new system being developed (i.e., to-be system), whether computerized or not. There are many different process modeling techniques in use today.³

A *use case* is a formal way of representing how a business system interacts with its environment. A use case illustrates the activities that are performed by the users of the system. As such, use case modeling is often thought of as an external or functional view of a business process in that it shows how the users view the process, rather than the internal mechanisms by which the process and supporting systems operate. Like activity diagrams, use cases can be used to document the current system (i.e., as-is system) or the new system being developed (i.e., to-be system).

Activity diagrams and use cases are *logical models*—models that describe the business domain's activities without suggesting how they are conducted. Logical models are sometimes referred to as problem domain models. When reading an activity diagram or use case, in principle, you should not be able to tell if an activity is computerized or manual; if a piece of information is collected by paper form or via the Web; if information is placed in a filing cabinet or a large database. These physical details are defined during the design phase, when the logical models are refined into *physical models*. These models provide information that is needed to ultimately build the system. By focusing on logical activities first, analysts can focus on how the business should run without being distracted with implementation details.

As a first step, the project team gathers requirements from the users (see Chapter 5). Next, using the gathered requirements, the project team models the overall business process using activity diagrams. Next the team uses the identified activities to identify the *use cases* that occur in the business. They then prepare *use case descriptions* and *use case diagrams* for each use case. Use cases are the discrete activities that the users perform, such as selling CDs, ordering CDs, and accepting returned CDs from customers. Users work closely with the project team to create the business process models in the form of activity diagrams and use

¹ Other similar techniques that are commonly used in non-UML projects are task modeling— see Ian Graham, *Migrating to Object Technology* (Reading, MA: Addison-Wesley, 1995), and Ian Graham, Brian Henderson-Sellers, and Houman Younessi, *The OPEN Process Specification*, (Reading, MA: Addison-Wesley, 1997)—and scenario-based design—see John M. Carroll, *Scenario-Based Design: Envisioning Work and Technology in System Development* (New York: John Wiley & Sons, 1995).

² We actually used an activity diagram to describe a simple process in Chapter 1 (see Figure 1-1).

³ Another commonly used process modeling technique is IDEF0. IDEF0 is used extensively throughout the U.S. federal government. For more information about IDEF0, see FIPS 183: Integration Definition for Function Modeling (IDEF0), Federal Information Processing Standards Publications, Washington, D.C.: U.S. Department of Commerce, 1993. Also, from an object-oriented perspective, a good book that utilizes the UML to address business process modeling is Hans-Erik Eriksson and Magnus Penker, *Business Modeling with UML* (New York: Wiley, 2000).

case descriptions that contain all the information needed to start modeling the system. Once the activity diagrams and use case descriptions are prepared, the systems analysts transform them into a use case diagram that portrays the external behavioral or functional view of the business process. Next, the analyst typically creates class diagrams (see the next chapter) to create a structural model of the business problem domain.

In this chapter, we first describe business process modeling and activity diagrams. Second, we describe use cases, their elements, and a set of guidelines for creating use cases. Third, we describe use case diagrams and how to create them. Fourth, using use cases as a basis, we revisit the topic of project size estimation.

BUSINESS PROCESS MODELING WITH ACTIVITY DIAGRAMS

Business process models describe the different activities that when combined together support a business process. Business processes typically cut across functional departments (e.g., the creation of a new product will involve many different activities that will combine the efforts of many employees in many departments). Furthermore, from an object-oriented perspective, they cut across multiple objects. As such, many of the earlier object-oriented systems development approaches tended to ignore business process modeling. Instead, they focused only on modeling processes via use cases (see later in this chapter) and behavioral models (see Chapter 8). However, today we realize that modeling business processes themselves is a very constructive activity that can be used to make sense out of the gathered requirements (see Chapter 5). The one potential problem of building business process models, from an object-oriented systems development perspective, is that they tend to reinforce a functional decomposition mindset. However, as long as they are used properly, they are a very powerful tool for communicating the analyst's current understanding of the requirements with the user. In this section of the chapter, we introduce the use of UML 2.0's activity diagrams as a means to build business process models.

Activity diagrams are used to model the behavior in a business process independent of objects.⁴ In many ways, activity diagrams can be viewed as sophisticated data flow diagrams that are used in conjunction with structured analysis. However, unlike data flow diagrams, Activity diagrams include notation that addresses the modeling of parallel, concurrent activities and complex decision processes. As such, activity diagrams can be used to model everything from a high-level business workflow that involve many different use cases, to the details of an individual use case, all the way down to the specific details of an individual method. In a nutshell, activity diagrams can be used to model any type of process.⁵ In this chapter, we restrict our coverage of activity diagrams to the modeling of high-level business processes.

Elements of an Activity Diagram

Activity diagrams portray the primary activities and the relationships among the activities in a process. Figure 6-1 shows the syntax of an activity diagram. Figure 6-2 presents a

⁴ For a good introduction to data flow diagrams and structured approaches to systems analysis and design, see Alan Dennis and Barbara Haley Wixom, *Systems Analysis Design*, 2nd Ed. (New York: Wiley, 2003).

⁵ Technically speaking, activity diagrams combine process modeling ideas from many different techniques including event models, statecharts, and Petri Nets. However, UML 2.0's activity diagram has more in common with Petri Nets than the other process modeling techniques. For a good description of using Petri Nets to model business workflows see Wil van der Aalst and Kees van Hee, *Workflow Management: Models, Methods, and Systems* (Cambridge, MA: MIT Press, 2002).

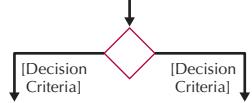
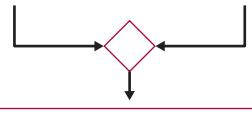
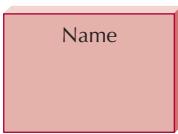
An Action:	
An Activity:	
An Object Node:	
A Control Flow:	
An Object Flow:	
An Initial Node:	
A Final-Activity Node:	
A Final-Flow Node:	
A Decision Node:	
A Merge Node:	
A Fork Node:	
A Join Node:	
A Swimlane:	

FIGURE 6-1 Syntax for an Activity Diagram

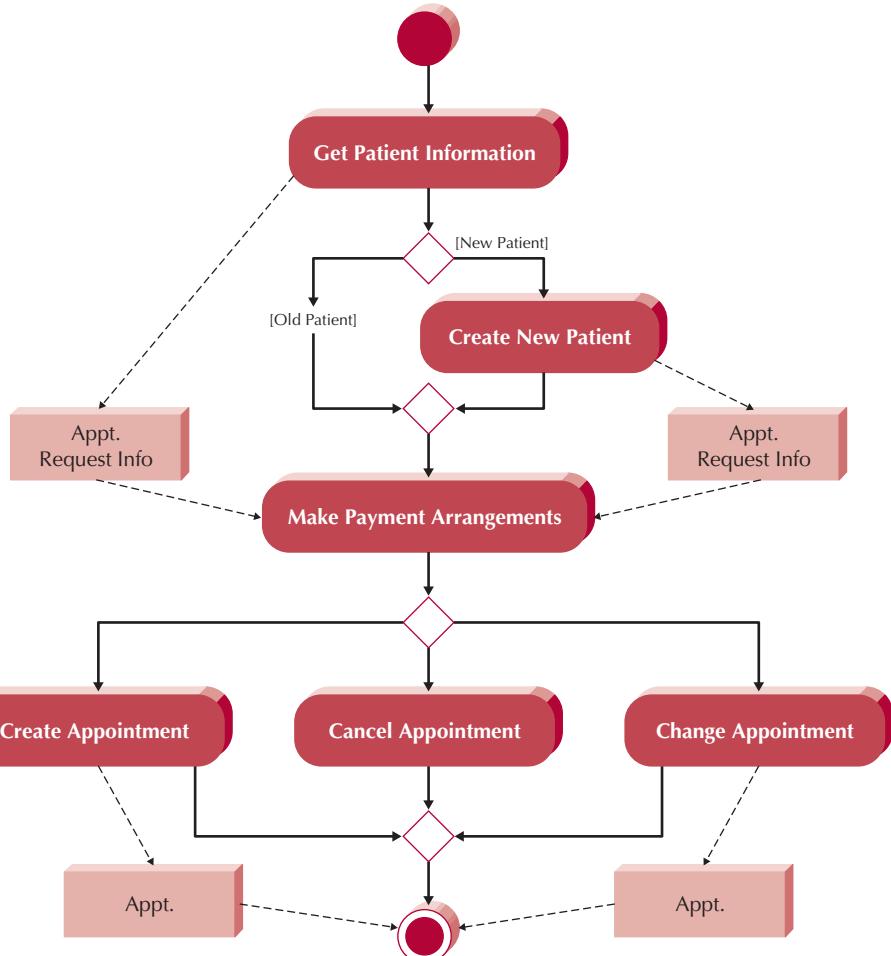


FIGURE 6-2
Activity Diagram for
Appointment System

simple activity diagram that represents part of an appointment system for a doctor's office.⁶ Next, we describe each of the different elements of an activity diagram.

Actions and Activities *Actions and activities* are performed for some specific business reason. Actions and activities can represent manual or computerized behavior. They are depicted in an activity diagram as a rounded rectangle (see Figure 6-1). Furthermore, they should have a name that begins with a verb and ends with a noun (e.g., “Make Appointment” or “Make Payment Arrangements”). Names should be short, yet contain enough information so that the reader can easily understand exactly what they do. The only difference between an action and an activity is that an activity can be decomposed further into a set of activities and/or actions, whereas an action represents a simple nondecomposable piece of the overall behavior being modeled. Typically, only activities are used for business

⁶ Due to the actual complexity of the syntax of activity diagrams, we follow a minimalist philosophy in our coverage (see John M. Carroll, *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill* (Cambridge, MA: MIT Press, 1990)). However, the material contained in this section is based on the *Unified Modeling Language: Superstructure Version 2.0*, ptc/03-08-02 (www.uml.org). Additional useful references include Michael Jesse Chonoles and James A. Schardt, *UML 2 for Dummies* (Indianapolis, IN: Wiley, 2003), Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado, *UML 2 Toolkit* (Indianapolis, IN: Wiley, 2004), and Kendall Scott, *Fast Track UML 2.0* (Berkeley, CA: Apress, 2004). For a complete description of all diagrams, see www.uml.org.

process or workflow modeling. Furthermore, in most cases, each activity will be associated with a use case. The activity diagram in Figure 6-2 shows six separate but related activities for a typical appointment system used in a doctor's office: Get Patient Information, Create New Patient, Make Payment Arrangements, Create Appointment, Cancel Appointment, and Change Appointment.

Object Nodes Activities and actions typically modify or transform objects. *Object nodes* model these objects in an activity diagram. Object nodes are portrayed in an activity diagram as a rectangle (see Figure 6-1). The name of the class of the object is written inside of the rectangle. Essentially, object nodes represent the flow of information from one activity to another activity. The simple appointment system portrayed in Figure 6-2 shows object nodes flowing from the Create Appointment and Change Appointment activities.

Control Flows and Object Flows There are two different types of “flows” in activity diagrams: control and object (see Figure 6-1). *Control flows* model the paths of execution through a business process. Control flows are portrayed as a solid line with an arrowhead on it showing the direction of flow. Control flows can only be attached to actions or activities. Figure 6-2 portrays a set of control flows through a doctor's office's appointment system. *Object flows* model the flow of objects through a business process. Since activities and actions modify or transform objects, object flows are necessary to show the actual objects that flow into and out of the actions or activities.⁷ Object flows are depicted as a dashed line with an arrowhead on it showing the direction of flow. An individual object flow must be attached to an action or activity on one end and an object node on the other end. Figure 6-2 portrays a set of control and object flows through the appointment system of a doctor's office.

Control Nodes There are seven different types of *control nodes* in an activity diagram: initial, final-activity, final-flow, decision, merge, fork, and join (see Figure 6-1). An *initial node* portrays the beginning of a set of actions or activities.⁸ An initial node is shown as a small filled in circle. A *final-activity node* is used to “stop the process” being modeled. Any time a final-activity node is reached, all actions and activities are ended immediately, regardless of whether they are completed. A final-activity node is represented as a circle surrounding a small filled-in circle making it resemble a bull's eye. A *final-flow node* is similar to a final-activity node, except that it stops a specific path of execution through the business process, but allows the other concurrent or parallel paths to continue. A final-flow node is shown as a small circle with an X in it.

The decision and merge nodes support modeling the decision structure of a business process. The *decision node* is used to represent the actual test condition that is used to determine which of the paths exiting the decision node is to be traversed. In this case, each of the exiting paths must be labeled with a guard condition. A *guard condition* represents the value of the test for that particular path to be executed. For example, in Figure 6-2, the decision node immediately below the “Get Patient Information” activity has two mutually exclusive paths that could be executed: one for old or previous patients, the other for new patients. The *merge node* is used to bring back together multiple mutually exclusive paths that have been split based on an earlier decision (e.g., the old and new patient paths in Figure 6-2 are brought back together).

⁷ For those familiar with data flow diagrams, these are similar to data flows.

⁸ For those familiar with IBM flowcharts, this is similar to the start node.

The fork and join nodes allow parallel and concurrent processes to be modeled (see Figure 6-1). The *fork node* is used to split the behavior of the business process into multiple parallel or concurrent flows. Unlike the decision node, the paths are not mutually exclusive (i.e., both paths are executed concurrently). For example, in Figure 6-3, the fork node is used to show that two concurrent, parallel processes are to be executed. In this case, each process is executed by two separate processors (parents). However, the purpose of the join node is similar to the merge node. The *join node* simply brings back together the separate parallel or concurrent flows in the business process into a single flow.

Swimlanes As already described, activity diagrams can be useful in modeling a business process independent of any object implementation. However, there are times that it is useful to break up an activity diagram in a manner that is useful in assigning responsibility to objects or individuals that would actually perform the activity. This is especially useful when modeling a business workflow. This is accomplished through the use of *swimlanes*. In Figure 6-3, the swimlanes are used to break up among two parents the making of a school lunch comprised of a peanut butter and jelly sandwich with a drink and dessert. In this case, we are using vertical swimlanes. You could also draw the activity diagram using more of a left-to-right orientation instead of a top-down orientation. In that case, the swimlanes would be drawn horizontally.

In an actual business workflow, we would see that we had activities that should be associated with roles of individuals that are involved in the business workflow (e.g., employees or customers) and the activities that were to be accomplished by the information system that was being created. This association of activities with external roles, internal roles, and the system is very useful when creating the use case descriptions and diagram described later in this chapter.

Guidelines for Creating Activity Diagrams

Scott Ambler has suggested the following guidelines when creating activity diagrams:⁹

1. Since an activity diagram can be used to model any kind of process, you should set the context or scope of the activity being modeled. Once you have determined the scope, you should give the diagram an appropriate title.
2. You must identify the activities, control flows, and object flows that occur between the activities.
3. You should identify any decisions that are part of the process being modeled.
4. You should attempt to identify any prospects for parallelism in the process.
5. You should draw the activity diagram.

When drawing an activity diagram, you should limit the diagram to a single initial node that starts the process being modeled. This node should be placed at the top or top-left of the diagram depending on the complexity of the diagram. Furthermore, for most business processes, there should only be a single final-activity node. This node should be placed at the bottom or bottom-right of the diagram (see Figures 6-2 and 6-3). Since most high-level business processes are sequential, not parallel, the use of a final-flow node should be limited.

⁹ For more details, see Scott W. Ambler, *The Object Primer: The Application Developer's Guide to Object Orientation and the UML, 2nd Ed.* (Cambridge University Press/SIGS Books, 2001) and Scott W. Ambler, *The Elements of UML Style* (Cambridge University Press, 2003).

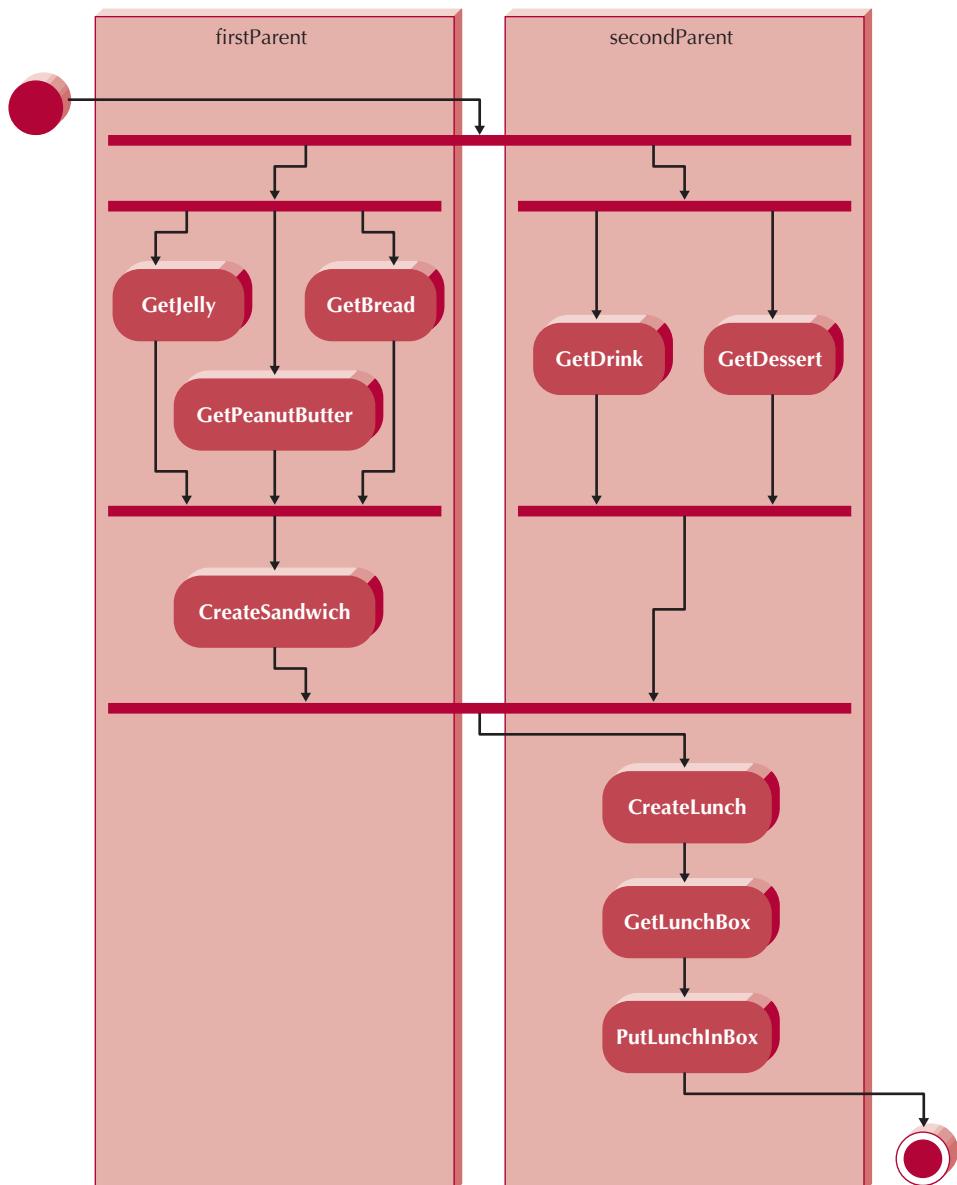


FIGURE 6-3
Activity Diagram for
Making a School Box
Lunch

When modeling high-level business processes or workflows, you should only include the more important decisions in the activity diagrams. In those cases, be sure that the guard conditions associated with the outflows of the decision nodes are mutually exclusive. Furthermore, the outflows and guard conditions should form a complete set (i.e., all potential values of the decision are associated with one of the flows).

As in decision modeling, you should only include forks and joins to represent the more important parallel activities in the process. For example, an alternative version of Figure 6-3 may not include the forks and joins associated with the GetJelly, GetBread, GetPeanutButter, GetDrink, and GetDessert activities. This would greatly simplify the diagram.¹⁰

¹⁰ In fact, the only reason we depicted the diagram in Figure 6-3 with the multiple fork and join nodes was to demonstrate that it could be done.

**YOUR
TURN**

6-1 Activity Diagrams

Look at the activity diagram for the appointment system in Figure 6-2. Think of one more activity that a user might ask for when gathering requirements for this system (e.g., maintaining patient insurance information). How would

you depict this on the existing diagram? After adding the activity to the diagram, what is your recommendation? Would you keep the new activity within the scope of this system? Why or why not?

When laying out the activity diagram, you should attempt to minimize line crossings to enhance the readability of the diagram. You also should lay out the activities on the diagram in a left to right and/or top to bottom order based on the order that the activities are executed. For example, in Figure 6-3, the Create Sandwich activity takes place before the Create Lunch activity.

Swimlanes should only be used to simplify the understanding of an activity diagram. Furthermore, the swimlanes should enhance the readability of the diagram. For example, when using a horizontal orientation for swimlanes, the top swimlane should represent the most important object or individual involved with the process. The order of the remaining swimlanes should be based on minimizing the number of flows crossing the different swimlanes. Also, when there are object flows among the activities associated with the different individuals (swimlanes) executing the activities of the process, it is useful to show the actual object flowing from one individual to another individual by including an object node “between” the two individuals (i.e., between the two swimlanes). This, of course, impacts how the swimlanes should be placed on the diagram.

Finally, you should challenge any activity that does not have any outflows or any inflows. Activities with no outflows are referred to as *black-hole activities*. If the activity is truly an end point in the diagram, the activity should have a control flow from it to a final-activity or final-flow node. An activity that does not have any inflow is known as a *miracle activity*. In this case, the activity is either missing an inflow from the initial node of the diagram or from another activity.

USE CASE DESCRIPTIONS

Use cases are simple descriptions of a system’s functions from the bird’s-eye view of the users.¹¹ Use case diagrams are functional diagrams in that they portray the basic functions of the system—that is, what the users can do and how the system should respond to the user’s actions.¹² Creating use case diagrams is a two-step process: First, the users work with the project team to write text-based *use case descriptions*, and second, the project team translates the use case descriptions into formal *use case diagrams*. Both the use case descriptions and the use case diagrams are based on the identified requirements and the activity diagram description of the

¹¹ For a more detailed description of use case modeling see Alistair Cockburn, *Writing Effective Use Cases* (Reading, MA: Addison-Wesley, 2001).

¹² Nonfunctional requirements, such as reliability requirements and performance requirements are often documented outside of the use case through more traditional requirements documents. See Gerald Kotonya and Ian Sommerville, *Requirements Engineering*, (Chichester, England: Wiley, 1998), Benjamin L. Kovitz, *Practical Software Requirements: A Manual of Content & Style* (Greenwich, CT: Manning, 1999), Dean Leffingwell and Donwidig, *Managing Software Requirements: A Unified Approach* (Reading, MA: Addison-Wesley, 2000), and Richard H. Thayer, M. Dorfman, and Sidney C. Bailin, Eds., *Software Requirements Engineering*, 2nd Ed. (IEEE Computer Society, 1997).

business process. Use case descriptions contain all the information needed to produce use case diagrams. Although it is possible to skip the use case description step and move directly to creating use case diagrams and the other diagrams that follow, users often have difficulty describing their business processes using only use case diagrams. Through the creation of use case descriptions, users can describe the required details of each individual use case. As to which should come first—use case descriptions or use case diagram—technically speaking, it really does not matter. Both should be done to fully describe the requirements that the information system must meet. In this text, we present the creation of the use case descriptions first.¹³

Use cases are the primary drivers for all of the UML diagramming techniques. The use case communicates at a high level what the system needs to do, and all of the UML diagramming techniques build on this by presenting the use case functionality in a different way for a different purpose. Use cases are the building blocks by which the system is designed and built.

Use cases capture the typical interaction of the system with the system's users (end users and other systems). These interactions represent the external, or functional, view of the system from the perspective of the user. Each use case describes one and only one function in which users interact with the system,¹⁴ although a use case may contain several "paths" that a user can take while interacting with the system (e.g., when searching for a book in Web bookstore, the user might search by subject, by author, or by title). Each path through the use case is referred to as a *scenario*.

When creating use cases, the project team must work closely with the users to gather the requirements needed for the use cases. This is often done through interviews, JAD sessions, and observation. Gathering the requirements needed for a use case is a relatively simple process, but it takes considerable practice. A good place to look for potential use cases is the activity diagram representation of the business process. In many cases, the activities identified in the activity diagram will become the use cases in the business process being modeled. The key thing to remember is that each use case is associated with one and only one *role* that users have in the system. For example, a receptionist in a doctor's office may "play" multiple roles—he or she can make appointments, answer the telephone, file medical records, welcome patients, and so on. Furthermore, it is possible that multiple users will play the same role. As such, use cases should be associated with the roles "played" by the users and not with the users themselves.

Types of Use Case

There are many different types of use cases. We suggest two separate dimensions on which to classify a use case based on the purpose of the use case and the amount of information that the use case contains: (1) overview versus detail; and (2) essential versus real.

¹³ Practically speaking, the analyst and users will iterate between the descriptions and diagram. You should not worry about which comes first. Whichever seems to work with the user community of the current project is the way that it should be done.

¹⁴ This is one key difference between traditional structured analysis and design approaches and object-oriented approaches. If you have experience using traditional structured approaches (or have taken a course on them) then this is an important change for you. If you have no experience with structured approaches then you can skip this footnote. The traditional structured approach is to start with one overall view of the system and to model processes via functional decomposition—the gradual decomposition of the overall view of the system into the smaller and smaller parts that make up the whole system. On the surface, this is similar to business process modeling using activity diagrams. However, functional decomposition is *not* used with object oriented approaches. Instead, each of the use cases documents one individual piece of the system; there is no overall use case that documents the entire system in the same way that a level 0 data flow diagram attempts to document the entire system. By removing this overall view, object oriented approaches make it easier to decouple the system's objects so they can be designed, developed and reused independently of the other parts of the system. While this lack of an overall view may prove unsettling initially, it is very liberating over the long term.

An *overview use case* is used to enable the analyst and user to agree on a high-level overview of the requirements. Typically, they are created very early in the process of understanding the system requirements, and they only document basic information about the use case such as its name, ID number, primary actor, type, and a brief description.

Once the user and the analyst agree upon a high-level overview of the requirements, the overview use cases can be converted to detail use cases. A *detail use case* typically documents, as far as possible, all of the information needed for the use case.

An *essential use case* is one that only describes the minimum essential issues necessary to understand the required functionality. A *real use case* will go further and describe a specific set of steps. For example, an essential use case in a dentist office might say that the receptionist should attempt to “Match the Patient’s desired appointment times with the available times,” while a real use case might say that the receptionist should “Look up the available dates on the calendar using MS Exchange to determine if the requested appointment times were available.” The primary difference is that essential use cases are implementation independent, while real use cases are detailed descriptions of how to use the system once it is implemented. As such, real use cases tend to be used only in detailed design, implementation, and testing.

Elements of a Use Case Description

A use case description contains all the information needed to build the diagrams that follow, but it expresses it in a less formal way that is usually simpler for users to understand. Figure 6-4 shows a sample use case description.¹⁵ There are three basic parts to a use case description: overview information, relationships, and the flow of events.

Overview Information The overview information identifies the use case and provides basic background information about the use case. The *use case name* of the use case should be a verb-noun phrase (e.g., Make Appointment). The *use case ID number* provides a unique way to find every use case and also enables the team to trace design decisions back to a specific requirement. As already stated, the *use case type* is either overview or detail and essential or real. The *primary actor* is usually the trigger of the use case—the person or thing that starts the execution of the use case. The primary purpose of the use case is to meet the goal of the primary actor. The *brief description* is typically a single sentence that describes the essence of the use case.

The *importance level* can be used to prioritize the use cases. As described in Chapter 2, object-oriented development tends to follow a RAD-phased development approach, in which some parts of the system are developed first and other parts are only developed in later versions. The importance level enables the users to explicitly prioritize which business functions are most important and need to be part of the first version of the system, and which are less important and can wait until later versions if necessary.

The importance level can use a fuzzy scale, such as high, medium, and low (e.g., in Figure 6-4 we have assigned an importance level of high to the Make Appointment use case). It can also be done more formally using a weighted average of a set of criteria. For example, Larman¹⁶ suggests rating each use case over the following criteria using a scale from zero to five:

¹⁵ Currently, there is no standard set of elements of a use case. The elements described in this section are based on recommendations contained in Alistair Cockburn, *Writing Effective Use Cases* (Reading, MA: Addison-Wesley, 2001); Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, 2nd Ed.* (Englewood Cliffs, NJ: Prentice-Hall, 2002); Brian Henderson-Sellers and Bhuvan Unhelkar, *OPEN modeling with UML* (Reading, MA: Addison-Wesley, 2000); Graham, *Migrating to Object Technology*; and Alan Dennis and Barbara Haley Wixom, *Systems Analysis and Design, 2nd Ed.* (New York: Wiley, 2003).

¹⁶ Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*.

- The use case represents an important business process.
- The use case supports revenue generation or cost reduction.
- Technology needed to support the use case is new or risky and therefore will require considerable research.

Use Case Name:	Make appointment	ID:	2	Importance Level:	High			
Primary Actor:	Patient	Use Case Type:	Detail, essential					
Stakeholders and Interests:								
Patient - wants to make, change, or cancel an appointment Doctor - wants to ensure patient's needs are met in a timely manner								
Brief Description: This use case describes how we make an appointment as well as changing or canceling an appointment.								
Trigger: Patient calls and asks for a new appointment or asks to cancel or change an existing appointment.								
Type: External								
Relationships:								
Association: Patient Include: Make Payment Arrangements Extend: Create New Patient Generalization:								
Normal Flow of Events:								
<ol style="list-style-type: none"> 1. The Patient contacts the office regarding an appointment. 2. The Patient provides the Receptionist with their name and address. 3. The Receptionist validates that the Patient exists in the Patient database. 4. The Receptionist executes the Make Payment Arrangements use case. 5. The Receptionist asks Patient if he or she would like to make a new appointment, cancel an existing appointment, or change an existing appointment. <ul style="list-style-type: none"> If the patient wants to make a new appointment, the S-1: new appointment subflow is performed. If the patient wants to cancel an existing appointment, the S-2: cancel appointment subflow is performed. If the patient wants to change an existing appointment, the S-3: change appointment subflow is performed. 6. The Receptionist provides the results of the transaction to the Patient. 								
Subflows:								
S-1: New Appointment <ol style="list-style-type: none"> 1. The Receptionist asks the Patient for possible appointment times. 2. The Receptionist matches the Patient's desired appointment times with available dates and times and schedules the new appointment. S-2: Cancel Appointment <ol style="list-style-type: none"> 1. The Receptionist asks the Patient for the old appointment time. 2. The Receptionist finds the current appointment in the appointment file and cancels it. S-3: Change Appointment <ol style="list-style-type: none"> 1. The Receptionist performs the S-2: cancel appointment subflow. 2. The Receptionist performs the S-1: new appointment subflow. 								
Alternate/Exceptional Flows:								
3a: The Receptionist executes the Create New Patient use case. S-1, 2a1: The Receptionist proposes some alternative appointment times based on what is available in the appointment schedule. S-1, 2a2: The Patient chooses one of the proposed times or decides not to make an appointment.								

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 6-4 Use Case Description Example

- Functionality described in the use case is complex, risky, and/or time critical. Depending on a use case's complexity, it may be useful to consider splitting its implementation over several different versions.
- The use case could increase understanding of the evolving design relative to the effort expended.

A use case may have multiple stakeholders that have an interest in the use case. As such, each use case lists each of the *stakeholders* with their interest in the use case (e.g., Patient and Doctor). The stakeholders' list always includes the primary actor (e.g., Patient).

Each use case typically has a *trigger*—the event that causes the use case to begin. For example, “Patient calls and asks for a new appointment or asks to cancel or change an existing appointment.” A trigger can be an *external trigger*, such as a customer placing an order or the fire alarm ringing, or it can be a *temporal trigger*, such as a book being overdue at the library, or time to pay the rent.

Relationships Use-case relationships explain how the use case is related to other use cases and users. There are four basic types of *relationships*: association, extend, include, and generalization. An *association relationship* documents the communication that takes place between the use case and the actors that use the use case. An actor is the UML representation for the *role* that a user plays in the use case. For example, in Figure 6-4, the Make Appointment use case is associated with the actor, Patient. In this case, a patient makes an appointment. All actors involved in the use case are documented with the association relationship.

An *extend relationship* represents the extension of the functionality of the use case to incorporate optional behavior. In Figure 6-4, the Make Appointment use case conditionally uses the Create New Patient use case. This use case is executed only if the patient does not exist in the patient database. As such, it is not part of the normal flow of events and should be modeled with an extend relationship and an alternate/exceptional flow.

An *include relationship* represents the mandatory inclusion of another use case. The include relationship enables functional decomposition—the breaking up of a complex use case into several simpler ones. For example, in Figure 6-4, the Make Payment Arrangements use case was considered to be complex and complete enough to be factored out as a separate use case that could be executed by the Make Appointment use case. The include relationship also enables parts of use cases to be reused by creating them as a separate use case.

The *generalization relationship* allows use cases to support *inheritance*. For example, the use case in Figure 6-4 could be changed such that a new patient could be associated with a specialized use case called Make New Patient Appointment and an old patient could be associated with a Make Old Patient Appointment. The common or generalized behavior that both the Make New Patient Appointment and Make Old Patient Appointment use cases contain would be placed in the generalized Make Appointment use case—for example, the include relationship to the Make Payment Arrangements use case. In other words, the Make New Patient Appointment and Make Old Patient Appointment use cases would *inherit* the Make Payment Arrangements use case from the Make Appointment use case. The specialized behavior would be placed in the appropriate specialized use case. For example, the extend relationship to the Create New Patient use case would be placed with the specialized Make New Patient Appointment use case as an included use case.

Flow of Events Finally, the individual steps within the business process are described. There are three different categories of steps or flows that can be documented: normal flow of events, subflows, and alternate or exceptional flows:

1. The *normal flow of events* includes only those steps that normally are executed in a use case. The steps are listed in the order in which they are performed. In Figure 6-4, the

Patient and the Receptionist have a conversation regarding the patient's name, address, and action to be performed.

2. In some cases, it is recommended to decompose the normal flow of events into a set of *subflows* to keep the normal flow of events as simple as possible. In Figure 6-4, we have identified three subflows: New Appointment, Cancel Appointment, and Change Appointment. Each of the steps of the subflows is listed. These subflows are based on the control flow logic in the activity diagram representation of the business process (see Figure 6-2). Alternatively, we could replace a subflow with a separate use case that could be included via the include relationships (see above). However, this should only be done if the newly created use case makes sense by itself. For example, in Figure 6-4, does it make sense to factor out a Create New Appointment, Cancel Appointment, and/or Change Appointment use case? If it does, then the specific subflow(s) should be replaced with a call to the related use case, and the use case should be added to the include relationship list.
3. *Alternate or exceptional flows* are ones that do happen but are not considered to be the norm. These must be documented. For example, in Figure 6-4, we have identified four alternate or exceptional flows. The first one simply addresses the need to create a new patient before the new patient can make an appointment. Normally, the patient would already exist in the patient database. Like the subflows, the primary purpose of separating out alternate or exceptional flows is to keep the Normal Flow of Events as simple as possible.

Optional Characteristics There are other characteristics of use cases that could be documented. These include the level of complexity of the use case, the estimated amount of time it takes to execute the use case, the system with which the use case is associated, specific data flows between the primary actor and the use case, any specific attribute, constraint, or operation associated with the use case, any preconditions that must be satisfied for the use case to execute, or any guarantees that can be made based on the execution of the use case. As we noted at the beginning of this section, there is no standard set of characteristics of a use case that must be captured. In this book, we suggest that the information contained in Figure 6-4 is the minimal amount to be captured.

Guidelines for Creating Use Case Descriptions

The essence of a use case is the *flow of events*. Writing the flow of events in a manner that is useful for later stages of development generally comes with experience. Figure 6-5 provides a set of guidelines that have proven to be useful.¹⁷

First, write each individual step in the form Subject-Verb-Direct Object and optionally Preposition-Indirect Object. This form has become known as *SVDPI* sentences. This form of sentence has proven to be useful in identifying classes and operations (see Chapter 7). For example, in Figure 6-4, the first step in the Normal Flow of Events, "The Patient contacts the office regarding an appointment," suggests the possibility of three classes of objects: Patient, Office, and Appointment. This approach simplifies the process of identifying the classes in the structural model (described in Chapter 7). Not all steps can use SVDPI sentences, but they should be used whenever possible.

Second, make clear who or what is the initiator of the action and who is the receiver of the action in each step. Normally, the initiator should be the subject of the sentence and the receiver should be the direct object of the sentence. For example, in Figure 6-4,

¹⁷ These guidelines are based on Cockburn, *Writing Effective Use Cases* and Graham, *Migrating to Object Technology*.

FIGURE 6-5
**Guidelines for Writing
 Effective Use Case
 Descriptions**

1. Write each set in the form of Subject-Verb-Direct Object (and sometimes Preposition-Indirect Object).
2. Make sure it is clear who the initiator of the step is.
3. Write the steps from the perspective of the independent observer.
4. Write each step at about the same level of abstraction.
5. Ensure the use case has a sensible set of steps.
6. Apply the KISS principle liberally.
7. Write repeating instructions after the set of steps to be repeated.

the second step, “Patient provides the Receptionist with their name and address,” clearly portrays the Patient as the initiator and the Receptionist as the receiver.

Third, write the step from the perspective of an independent observer. To accomplish this, you may have to write each step from the perspective of both the initiator and the receiver first. Based on the two points of view, you can then write the bird’s eye view version. For example, in Figure 6-4, the “Patient provides the Receptionist with their name and address.” Neither the patient’s nor the receptionist’s perspective is represented.

Fourth, write each step at the same level of abstraction. Each step should make about the same amount of progress toward completing the use case as each of the other steps in the use case. On high-level use cases, the amount of progress could be very substantial, while in a low-level use case, each step could represent only incremental progress. For example, in Figure 6-4, each step represents about the same amount of effort to complete.

Fifth, ensure that the use case contains a sensible set of actions. Each use case should represent a transaction. Therefore, each use case should be comprised of four parts:

1. The primary actor initiates the execution of the use case by sending a request (and possibly data) to the system.
2. The system ensures that the request (and data) is valid.
3. The system processes the request (and data) and possibly changes its own internal state.
4. The system sends the primary actor the result of the processing.

For example, in Figure 6-4, (1) the patient requests an appointment (Steps 1 and 2), (2) the receptionist determines if the patient exists in the database (Step 3), (3) the receptionist sets up the appointment transaction (Steps 4 and 5), and (4) the receptionist gives the time and date of the appointment to the patient (Step 6).

The sixth guideline is the KISS¹⁸ principle. If the use case becomes too complex and/or too long, the use case should be decomposed into a set of use cases. Furthermore, if the Normal Flow of Events of the use case becomes too complex, subflows should be used. For example, in Figure 6-4, the fifth step in the Normal Flow of Events was sufficiently complex to decompose it into three separate subflows. However, as stated previously, care must be taken to avoid the possibility of decomposing too much. Most decomposition should be done with classes (see Chapter 7).

The seventh guideline deals with repeating steps. Normally, in a programming language such as Visual Basic or C, we put loop definition and controls at the beginning of the loop. However, since the steps are written in simple English, it is normally better to simply write “Repeat steps A through E until some condition is met” after step E. It makes the use case more readable to people unfamiliar with programming.

¹⁸ Keep It Simple, Stupid.

**YOUR
TURN****6-2 Use Cases**

Look at the activity diagram for the appointment system in Figure 6-2 and the use case that was created in Figure 6-4. Create your own use case based on an activity in the

activity diagram or the activity that you created in Your Turn 6-1. Use Figure 6-5 to guide your efforts.

USE CASE DIAGRAMS

In the previous section, we learned what a use case is and discussed a set of guidelines on how to write effective use case descriptions. In this section, we will learn how the use case is the building block for the *use case diagram*, which summarizes all of the use cases for the part of the system being modeled together in one picture. An analyst can use the use case diagram to better understand the functionality of the system at a very high level. Typically, the use case diagram is drawn early on in the SDLC when gathering and defining requirements for the system because it provides a simple, straightforward way of communicating to the users exactly what the system will do. In this manner, the use case diagram can encourage the users to provide additional requirements that the written use case may not uncover. This section describes the elements of a use case diagram.

A use case diagram illustrates in a very simple way the main functions of the system and the different kinds of users that will interact with it. Figure 6-6 describes the basic syntax rules for a use case diagram. Figure 6-7 presents a use case diagram for a doctor's office appointment system. We can see from the diagram that patients, doctors, and management personnel will use the appointment system to make appointments, record availability, and produce schedule information, respectively.

Actor

The labeled stick figures on the diagram represent actors (see Figure 6-6). An *actor* is not a specific user, but a role that a user can play while interacting with the system. An actor can also represent another system in which the current system interacts. In this case, the actor optionally can be represented by a rectangle with <<actor>> and the name of the system. Basically, actors represent the principal elements in the environment in which the system operates. Actors can provide input to the system, receive output from the system, or both. The diagram in Figure 6-7 shows that three actors will interact with the appointment system (a patient, a doctor, and management).

Sometimes an actor plays a specialized role of a more general type of actor. For example, there may be times when a new patient interacts with the system in a way that is somewhat different than a general patient. In this case, a *specialized actor* (i.e., new patient) can be placed on the model, shown using a line with a hollow triangle at the end of the more general actor (i.e., patient). The specialized actor will inherit the behavior of the more general actor and extend it in some way (see Figure 6-8). Can you think of some ways in which a new patient may behave differently than an existing patient?

Association

Use cases are connected to actors through *association relationships*, which show with which use cases the actors interact (see Figure 6-6). A line drawn from an actor to a use

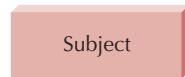
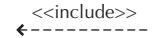
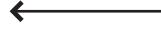
An Actor:	<ul style="list-style-type: none"> ■ Is a person or system that derives benefit from and is external to the subject ■ Is depicted as either a stick figure (default) or if a non-human actor is involved, as a rectangle with <> in it (alternative) ■ Is labeled with its role ■ Can be associated with other actors using a specialization/superclass association, denoted by an arrow with a hollow arrowhead ■ Are placed outside the subject boundary 	 <> Actor/Role
A Use Case:	<ul style="list-style-type: none"> ■ Represents a major piece of system functionality ■ Can extend another use case ■ Can include another use case ■ Is placed inside the system boundary ■ Is labeled with a descriptive verb-noun phrase 	
A Subject Boundary:	<ul style="list-style-type: none"> ■ Includes the name of the subject inside or on top ■ Represents the scope of the subject, e.g., a system or an individual business process 	
An Association Relationship:	<ul style="list-style-type: none"> ■ Links an actor with the use case(s) with which it interacts 	
An Include Relationship:	<ul style="list-style-type: none"> ■ Represents the inclusion of the functionality of one use case within another ■ The arrow is drawn from the base use case to the included use case 	
An Extend Relationship:	<ul style="list-style-type: none"> ■ Represents the extension of the use case to include optional behavior ■ The arrow is drawn from the extension use case to the base use case 	
A Generalization Relationship:	<ul style="list-style-type: none"> ■ Represents a specialized use case to a more generalized one ■ The arrow is drawn from the specialized use case to the base use case 	

FIGURE 6-6 Syntax for Use Case Diagram

case depicts an association. The association typically represents two-way communication between the use case and the actor. If the communication is only one way, then a solid arrowhead can be used to designate the direction of the flow of information. For example, in Figure 6-7 the Patient actor communicates with the Make Appointment use case. Since there are no arrowheads on the association, the communication is two-way. Finally, it is possible to represent the multiplicity of the association. Figure 6-7 shows an “*” at either end of the association between the Patient and the Make Appointment use case. This simply designates that an individual patient (instance of the Patient actor) executes

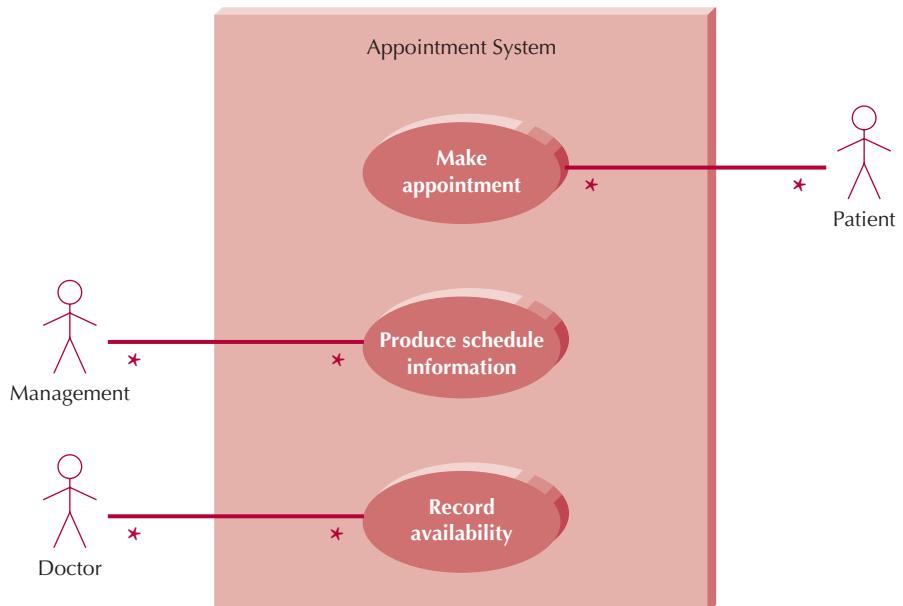


FIGURE 6-7
Use Case Diagram for
Appointment System

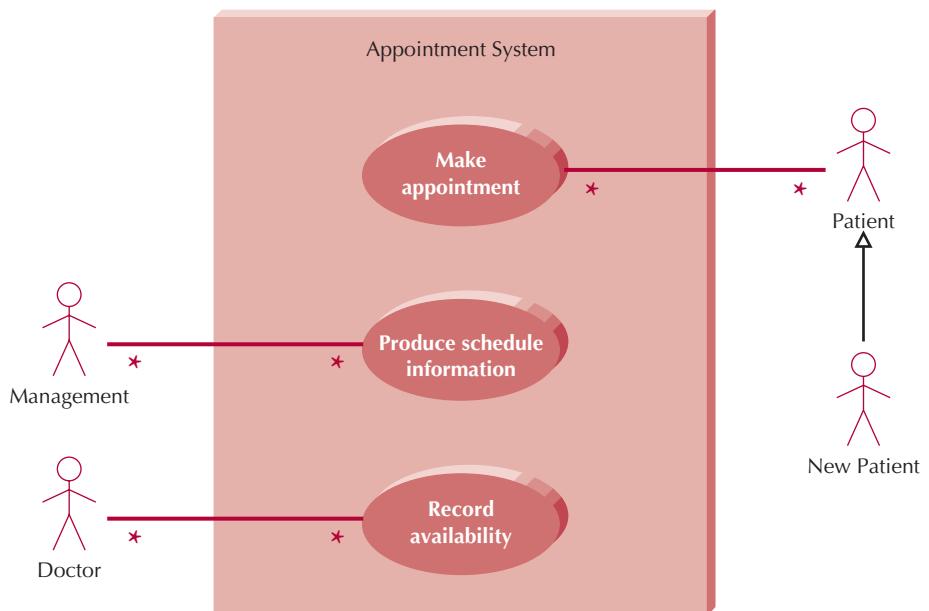


FIGURE 6-8
Use Case Diagram with
Specialized Actor

the Make Appointment use case as many times as they wish and that it is possible for the Make Appointment use case appointment to be executed by many different patients. In most cases, this type of many-to-many relationship is appropriate. However, it is possible to restrict the number of patients that can be associated with the Make Appointment use case. We will discuss the multiplicity issue in detail in the next chapter in regards to class diagrams.

Use Case

A *use case*, depicted by an oval in the UML, is a major process that the system will perform that benefits an actor(s) in some way (see Figure 6-6), and it is labeled using a descriptive verb-noun phrase. We can tell from Figure 6-7 that the system has three primary use cases: Make Appointment, Produce Schedule Information, and Record Availability.

There are times when a use case includes, extends, or generalizes the functionality of another use case on the diagram. These are shown using include, extend, and generalization relationships. To increase the ease of understanding a use case diagram, “higher-level” use cases normally are drawn above the “lower-level” ones. It may be easier to understand these relationships with the help of examples. Let’s assume that every time a patient makes an appointment, the patient is asked to verify payment arrangements. However, occasionally it is necessary to actually make new payment arrangements. Therefore, we may want to have a use case called Make Payment Arrangements that *extends* the Make Appointment use case to include this additional functionality. In Figure 6-9, an arrow labeled with extend

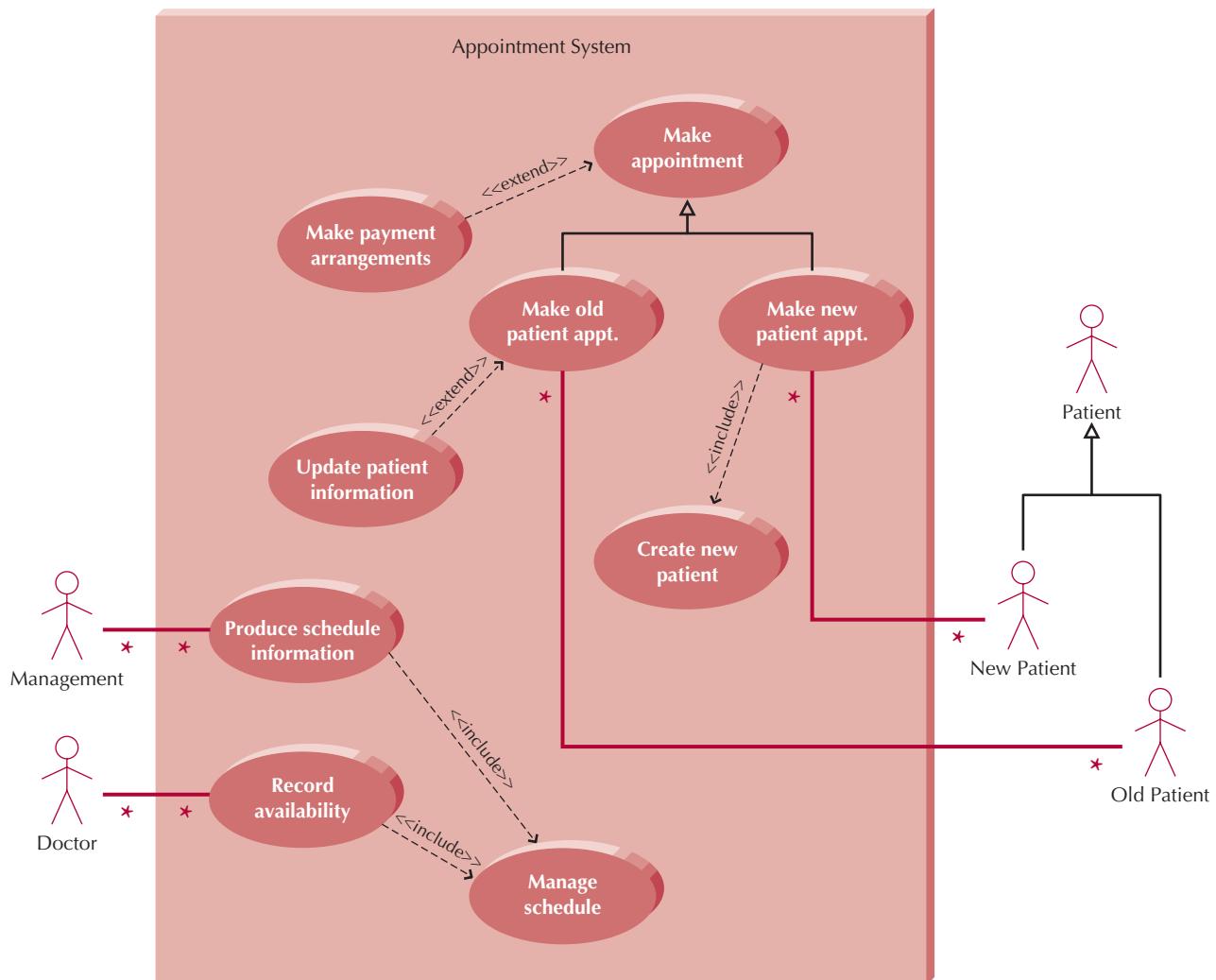


FIGURE 6-9 Extend and Include Relationships

was drawn between Make Payment Arrangements use case and the Make Appointment use case to denote this special use case relationship. Furthermore, the Make Payment Arrangements use case was drawn lower than the Make Appointment use case.

Similarly, there are times when a single use case contains common functions that are used by other use cases. For example, suppose there is a use case called Manage Schedule that performs some routine tasks needed to maintain the doctor's office appointment schedule, and the two use cases Record Availability and Produce Schedule Information both perform the routine tasks. Figure 6-9 shows how we can design the system so that Manage Schedule is a shared use case that is used by others. An arrow labeled with include is used to denote the include relationship and the *included* use case is drawn below the use cases that contain it.

Finally, there are times that it makes sense to use a generalization relationship to simplify the individual use cases. For example in Figure 6-9, the Make Appointment use case has been specialized to include a use case for an Old Patient and a New Patient. The Make Old Patient Appt use case inherits the functionality of the Make Appointment use case (including the Make Payment Arrangements use case extension) and extends its own functionality with the Update Patient Information use case. The Make New Patient Appt use case also inherits all of the functionality of the generic Make Appointment use case and calls the Create New Patient use case, which includes the functionality necessary to insert the new patient into the patient database. The generalization relationship is represented as an unlabeled hollow arrow with the more general use case being higher than the lower-use cases. Also notice that we have added a second specialized actor, Old Patient, and that the Patient actor is now simply a generalization of the Old and New Patient actors.

Subject Boundary

The use cases are enclosed within a *subject boundary*, which is a box that defines the scope of the system and clearly delineates what parts of the diagram are external or internal to it (see Figure 6-6). One of the more difficult decisions to make is where to draw the subject boundary. A subject boundary can be used to separate a software system from its environment, a subsystem from other subsystems within the software system, or an individual process in a software system. They also can be used to separate an information system, including both software and internal actors, from its environment. As such, care should be taken to carefully decide on what the scope of the information system is to include.

The name of the subject can appear either inside or on top of the box. The subject boundary is drawn based on the scope of the system. In the appointment system, the Management and Doctor actors may not be drawn. Remember that actors are outside of the scope of the system. In Figure 6-7, we listed a receptionist as being part of the use case. However, in this case, we assumed that the receptionist is an "internal" actor that is part of the Make Appointment use case. As such, the receptionist is not drawn on the diagram.¹⁹

CREATING USE CASE DESCRIPTIONS AND USE CASE DIAGRAMS

Use cases are used to describe the functionality of the system and as a model of the dialog between the actors and the system. As such, they tend to be used to model both the contexts of the system and the detailed requirements for the system. Even though the primary

¹⁹ In other non-UML approaches to object-oriented systems development, it is possible to represent external actors along with "internal" actors. In this example, the receptionist would be considered an internal actor (see Graham, *Migrating to Object Technology*, and Ian Graham, Brian Henderson-Sellers, and Houman Younessi, *The OPEN Process Specification*).

purpose of use cases is to document the functional requirements of the system, they also are used as a basis for testing the evolving system. In this section, we describe an approach that supports requirements gathering and documentation with use cases.

It is important to remember that use cases are used for both the as-is and to-be behavioral models. As-is use cases focus on the current system, while to-be use cases focus on the desired new system. The two most common ways to gather information for the use cases are through interviews and JAD sessions (observation is also sometimes used for as-is models). As discussed in Chapter 5, these techniques have advantages and disadvantages.

Regardless of whether interviews or JAD session are used, recent research shows that some ways to gather the information for use cases are better than others. The most effective process has thirteen steps to create use case descriptions²⁰ and four additional steps to create use case diagrams (see Figure 6-10). These thirteen steps are performed in order, but of course the analyst often cycles among them in an iterative fashion as he or she moves from use case to use case.

Identify the Major Use Cases

The first step is to review the activity diagram. This helps the analyst to get a complete overview of the underlying business process being modeled.

Identify the Major Use Cases

1. Review the activity diagram.
2. Find the subject's boundaries.
3. Identify the primary actors and their goals.
4. Identify and write the overviews of the major use cases for the above.
5. Carefully review the current use cases. Revise as needed.

Expand the Major Use Cases

6. Choose one of the use cases to expand.
7. Start filling in the details of the chosen use case.
8. Write the Normal Flow of Events of the use case.
9. If the Normal Flow of Events is too complex or long, decompose into subflows.
10. List the possible alternate or exceptional flows.
11. For each alternate or exceptional flow, list how the actor and/or system should react.

Confirm the Major Use Cases

12. Carefully review the current set of use cases. Revise as needed.
13. Start at the top again.

Create the Use Case Diagram

1. Draw the subject boundary.
2. Place the use cases on the diagram.
3. Place the actors on the diagram.
4. Draw the associations.

FIGURE 6-10
Steps for Writing
Effective Use-Case
Descriptions and Use-
Case Diagrams

²⁰ The approach in this section is based on the work of Alistair Cockburn, *Writing Effective Use Cases*; Graham, *Migrating to Object Technology*; George Marakas and Joyce Elam, "Semantic Structuring in Analyst Acquisition and Representation of Facts in Requirements Analysis," *Information Systems Research* 9(1) (1998), 37-63 as well as our own: Alan Dennis, Glenda Hayes, and Robert Daniels, "Business Process Modeling with Group Support Systems," *Journal of MIS* 15(4) (1999), 115-142.

The second step is to identify the subject's boundaries. This helps the analyst to identify the scope of the system. However, as we work through the SDLC, the boundary of the system will change.

The third step is to identify the primary actors and their goals. The primary actors involved with the system will come from a list of stakeholders and users. However, you should remember that an actor is a role that a stakeholder or user plays, not a specific user (e.g., doctor, not Dr. Jones). The goals represent the functionality that the system must provide the actor for the system to be a success. Identifying what the tasks are that each actor must perform can facilitate this. For example, does the actor need to create, read, update, or delete (CRUD) any information currently in the system, are there any external changes that an actor must inform the system, or is there any information that the system should notify the actor? Steps two and three are intertwined. As actors are identified and their goals are uncovered, the boundary of the system will change.

The fourth step is to identify and write the major use cases, with basic information about each, rather than jumping into one use case and describing it completely (i.e., only an overview use case). Recall from the previous description of the elements of a use case that overview use cases only contain the use case name, ID number, type, primary actor, importance level, and brief description. Creating only overview use cases at this time prevents the users and analysts from forgetting key use cases and helps the users explain the overall set of business processes for which they are responsible. It also helps them understand how to describe the use cases and reduces the chance of overlap between use cases. It is important at this point to understand and define acronyms and jargon so that the project team and others from outside the user group can clearly understand the use cases. Again, the activity diagram is a very useful beginning point for this step.

The fifth step is to carefully review the current set of use cases. It may be necessary to split some of them into multiple use cases or merge some of them into a single use case. Also, based on the current set, a new use case may be identified. You should remember that identifying use cases is an iterative process, with users often changing their minds about what is a use case and what it includes. It is very easy to get trapped in the details at this point, so you need to remember that the goal at this step is to identify the *major* use cases and that you are only creating overview use cases. For example, in the doctor's office example in Figure 6-9, we defined one use case as Make Appointment. This use case included the cases for both new patients and existing patients, as well as when the patient changes or cancels the appointment. We could have defined each of these activities (makes an appointment, changes an appointment, or cancels an appointment) as separate use cases, but this would have created a huge set of "small" use cases.

The trick is to select the right "size" so that you end up with three to nine use cases in each system. If the project team discovers many more than eight use cases, this suggests that the use cases are "too small" or that the system boundary is too big. There should be no more than three to nine use cases on the model, so if you identify more than nine, you should group together the use cases into *packages* (i.e., logical groups of use cases) to make the diagrams easier to read and keep the models at a reasonable level of complexity. It is simple at that point to sort the use cases and group together these "small" use cases into "larger" use cases that include several small ones or to change the system boundaries.²¹

Expand the Major Use Cases

The sixth step is to choose one of the major use cases to expand. Using the importance level of the use case can do this. For example, in Figure 6-4, the Make Appointment use case has

²¹ For those familiar with structured analysis and design, packages serve a similar purpose as the leveling and balancing processes used in data flow diagramming. Packages are described in Chapter 9.

an importance level of high. As such, it would be one of the earlier use cases to be expanded. You could also use the criteria suggested by Larman²² to set the prioritization of the use cases as noted earlier.

The seventh step is to start filling in the details on the use case template. For example, list all of the identified stakeholders and their interests in the use case, the level of importance of the use case, brief description of the use case, the trigger information for the use case, and the relationships in which the use case participates.

The eighth step is to fill in the steps of the normal flow of events required to describe each use case. The steps focus on what the business process does to complete the use case, as opposed to what actions the users or other external entities do. In general, the steps should be listed in the order in which they are performed, from first to last. Remember to write the steps in an SVDPI form whenever possible.

The ninth step is to ensure that the steps listed in the normal flow of events are not too complex or too long. Each step should be about the same size as the others. For example, if we were writing steps for preparing a meal, steps such as “take fork out of drawer” and “put fork on table” are much smaller than “prepare cake using mix.” If you end up with more than seven steps or steps that vary greatly in size, you should go back and review each step carefully and possibly rewrite the steps.

One good approach to produce the steps for a use case is to have the users visualize themselves actually performing the use case and to have them write down the steps as if they were writing a recipe for a cookbook. In most cases the users will be able to quickly define what they do in the as-is model. Defining the steps for to-be use cases may take a bit more coaching. In our experience, the descriptions of the steps change greatly as users work through a use case. Our advice is to use a blackboard or whiteboard (or paper with pencil) that can be easily erased to develop the list of steps, and then write it on the use case form. You should write it on the use case form only after the set of steps is fairly well defined.

The tenth step focuses on identifying the alternate or exceptional flows. Alternate or exceptional flows are those flows of success that represent optional or exceptional behavior. They tend to occur infrequently or as a result of a normal flow failing. They should be labeled in a way that there is no doubt as to which normal flow of event to which it is related. For example in Figure 6-4, Alternate/Exceptional flow 3a executes when step 3 fails (i.e., the Patient does not exist in the Patient database).

The eleventh step is simply to write the description of the alternate or exceptional flow. In other words, if the alternate or exceptional flow is to be executed, describe the response that the actor or system should produce. Like the normal flows and subflows, alternate/exceptional flows should be written in the SVDPI form whenever possible.

Confirm the Major Use Cases

The twelfth step is to carefully review the current set of use cases and to confirm that the use case is correct as written, which means reviewing the use case with the users to make sure each step is correct. The review should look for opportunities to simplify a use case by decomposing it into a set of smaller use cases, merging it with others, looking for common aspects in both the semantics and syntax of the use cases, and identifying new use cases. This is also the time to look into adding the include, extend, or generalization relationships between use cases. The most powerful way to confirm the use case is to ask the user to role play, or execute the process using the written steps in the use case. The analyst will hand the user pieces of paper labeled as the major inputs to the use case, and will have the user follow the written steps like a recipe to make sure that those steps really can produce the outputs defined for the use case using its inputs.

²² Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*.

The thirteenth and final step is to *iterate* over the entire set of steps again. Users will often change their minds about what is a use case and what it includes. It is very easy to get trapped in the details at this point, so you need to remember that the goal is to just address the major use cases. Therefore, you should continue to iterate over these steps until you and the users believe that a sufficient number of use cases have been documented to begin identifying candidate classes for the structural model (see next chapter). As you identify candidate classes, it is likely that additional use cases will be uncovered. Remember, object-oriented systems development is both iterative and incremental. As such, you should not worry about identifying each and every possible use case at this point in the development of the system.

Create the Use Case Diagram

Basically, drawing the use case diagram is very straightforward once you have detailed use cases. The actual use case diagram encourages the use of information hiding. The only information that is drawn on the use case diagram are the system boundary, the use cases themselves, the actors, and the various associations between these components. The major strength of the use case diagram is that it provides the user with an overview of the detailed use cases. However, you must remember that anytime a use case changes, it could impact the use case diagram.

There are four major steps to drawing the use case diagram. First, the use case diagram starts with the subject boundary. This forms the border of the subject; separating use cases (i.e., the subject's functionality) from actors (i.e., the roles of the external users).

Second, the use cases are drawn on the diagram. These are taken directly from the detailed use case descriptions. Special use case associations (include, extend, or generalization) are also added to the model at this point. Be careful in laying out the diagram. There is no formal order to the use cases so they can be placed in whatever fashion is needed to make the diagram easy to read and to minimize the number of lines that cross. It often is necessary to redraw the diagram several times with use cases in different places to make the diagram easy to read. Also, for understandability purposes, there should be no more than three to nine use cases on the model to the diagram as simple as possible. This includes those use cases that have been factored out and now are associated with another use case through the include, extend, or generalization relationships.

Third, the actors are placed on the diagram. The actors are taken directly from the primary actor element on the detailed use case description. Like use case placement, to minimize the number of lines that will cross on the diagram, the actors should be placed near the uses cases with which they are associated.

The fourth and last step is to draw lines connecting the actors to the use cases with which they interact. No order is implied by the diagram, and the items you have added along the way do not have to be placed in a particular order; therefore, you may want to rearrange the symbols a bit to minimize the number of lines that cross so the diagram is less confusing.

REFINING PROJECT SIZE AND EFFORT ESTIMATION USING USE CASE POINTS²³

Back in Chapter 4, we used function points and the COCOMO method to estimate the size and effort of a systems development project, respectively. However, neither of these approaches were developed nor validated with object-oriented systems in mind. As such,

²³ The material in this section is based on descriptions of use case points contained in Raul R. Reed, Jr., *Developing Applications with Java and UML* (Reading, MA: Addison-Wesley, 2002), Geri Schneider and Jason P. Winters, *Applying Use Cases: A Practical Guide* (Reading, MA: Addison-Wesley, 1998), and Kirsten Ribu, *Estimating Object-Oriented Software Projects with Use Cases*, Masters Thesis, Department of Informatics, University of Oslo, 2001.

**YOUR
TURN**

6-3 Use Case Diagram

Look at the Use Case Diagram in Figure 6-9. Consider if a use case were added to maintain patient insurance infor-

mation. Make assumptions about the details of this use case and add it to the existing use case diagram in Figure 6-9.

even though they are very popular approaches to estimation, their application to object-oriented systems is questionable. Since you now know about use cases, we introduce a size and effort *estimation* technique that was developed around their use. This technique, known as use case points, was originally developed by Gustav Karner of Objectory AB.²⁴ *Use case points* are based on the same ideas that function points were developed. However, they have been refined to take into consideration of the unique features of use cases and object orientation.

To estimate size and effort using use case points, first you must have created at least the set of essential use cases and the use case diagram. Otherwise, the information required will not be available. Once the use cases and use case diagram have been created, you need to classify the actors and use cases as being simple, average, or complex.

Simple actors are separate systems that the current system must communicate through a well-defined *application program interface (API)*. *Average actors* are separate systems that interact with the current system using standard communication protocols, such as TCP/IP, FTP, or HTTP, or an external database that can be accessed using standard SQL. *Complex actors* are typically end users communicating with the system using some type of GUI. Once all of the actors have been classified, the appropriate numbers should be entered into the Unadjusted Actor Weighting Table contained in the Use Case Point estimation worksheet (see Figure 6-11). For example, reviewing the Make Appointment use case (see Figure 6-4) and the use case diagram for the Appointment system (see Figure 6-9) we see that we have four human actors that interact with the system giving an *Unadjusted Actor Weight Total (UAW)* of 12.

Depending on the number of unique transactions that the use case must address, a use case is classified as a *simple use case* (1–3), an *average use case* (4–7), or a *complex use case* (>7). In the original formulation of use case point estimation, Karner suggested that included and extended use cases should be ignored. However, today the recommendation is to use all use cases, regardless of their type in the estimation calculation. For example, the Make Appointment use case described in Figure 6-4 must deal with a set of activities including making a new appointment, canceling an existing appointment, changing an existing appointment, creating a new patient, and making payment arrangements. In this case, since the Make Appointment use case has to deal with five separate transactions, it would be classified as an average use case (see the Unadjusted Use Case Weighting Table in Figure 6-11). Based on the Appointment System use case diagram (see Figure 6-9), we have three simple use cases (Produce Schedule, Make Old Patient Appointment, and Record Availability), four average use cases (Make Appointment, Make New Patient Appointment, Manage Schedule, and Make Payment Arrangements), and one complex use case (Update Patient Information) giving an *Unadjusted Use Case Weight Total (UUCW)* of 70.

²⁴ Objectory AB was acquired by Rational back in 1995. Rational is now part of IBM.

Unadjusted Actor Weighting Table:				
Actor Type	Description	Weighting Factor	Number	Result
Simple	External System with well-defined API	1		
Average	External System using a protocol-based interface, e.g., HTTP, TCT/IP, or a database	2		
Complex	Human	3		
Unadjusted Actor Weight Total (UAW)				
Unadjusted Use Case Weighting Table:				
Use Case Type	Description	Weighting Factor	Number	Result
Simple	1–3 transactions	5		
Average	4–7 transactions	10		
Complex	>7 transactions	15		
Unadjusted Use Case Weight Total (UUCW)				
Unadjusted Use Case Points (UUCP) = UAW + UUCW				
Technical Complexity Factors:				
Factor Number	Description	Weight	Assigned Value (0–5)	Weighted Value
T1	Distributed system	2.0		
T2	Response time or throughput performance objectives	1.0		
T3	End-user online efficiency	1.0		
T4	Complex internal processing	1.0		
T5	Reusability of code	1.0		
T6	Easy to install	0.5		
T7	Ease of use	0.5		
T8	Portability	2.0		
T9	Ease of change	1.0		
T10	Concurrency	1.0		
T11	Special security objectives included	1.0		
T12	Direct access for third parties	1.0		
T13	Special User training required	1.0		
Technical Factor Value (TFactor)				
Technical Complexity Factor (TCF) = 0.6 + (0.01 * TFactor)				
Environmental Factors:				
Factor Number	Description	Weight	Assigned Value (0 – 5)	Weighted Value
E1	Familiarity with system development process being used	1.5		
E2	Application experience	0.5		
E3	Object-oriented experience	1.0		
E4	Lead analyst capability	0.5		
E5	Motivation	1.0		
E6	Requirements stability	2.0		
E7	Part time staff	-1.0		
E8	Difficulty of programming language	-1.0		
Environmental Factor Value (EFactor)				
Environmental Factor (EF) = 1.4 + (-0.03 * EFactor)				
Adjusted Use Case Points (UCP) = UUCP * TCF * ECF				
Effort in Person Hours = UCP * PHM				

TEMPLATE
can be found at
[www.wiley.com/
college/dennis](http://www.wiley.com/college/dennis)

FIGURE 6-11 Use Case Point Estimation Worksheet

The value of the *Unadjusted Use Case Points (UUCP)* is simply the sum of the Unadjusted Actor Weight Total (12) and the Unadjusted Use Case Weight Total (70). As such, UUCP is equal to 82.

In the spirit of function point analysis, use case point-based estimation also has a set of factors that are used to adjust the use case point value. In this case, there are two sets of factors: *technical complexity factors (TCF)* and *environmental factors (EF)*. There are thirteen separate technical factors and eight separate environmental factors. The purpose of these factors is to allow the project as a whole to be evaluated for complexity and experience levels of the staff, respectively. Obviously, these types of factors can impact the effort required by a team to develop a system. Each of these factors are assigned a value between 0 and 5; 0 representing that the factor is irrelevant to the system under consideration and 5 representing that the factor is essential for the system to be successful. The assigned values are then multiplied by their respective weights. These weighted values are then summed up to create a *technical factor value (TFactor)* and an *environmental factor value (EFactor)*.

The technical factors include the following:

- Whether the system was going to be a distributed system
- The importance of response time
- The efficiency level of the end user using the system
- The complexity of the internal processing of the system
- The importance of code reuse
- How easy the installation process had to be
- The importance of the ease of using the system
- How important it was for the system to be able to be ported to another platform
- Whether system maintenance is important
- Whether the system is going to have to handle parallel and concurrent processing
- The level of special security required
- The level of system access by third parties
- Whether special end user training was to be required (see Figure 6-11)

The environmental factors include the following:

- The level of experience the development staff has with the development process being used
- The application being developed
- The level of object-oriented experience
- The level of capability of the lead analyst
- The level of motivation of the development team to deliver the system
- The stability of the requirements
- Whether part-time staff have to be included as part of the development team
- The difficulty of the programming language being used to implement the system (see Figure 6-11)

Continuing the appointment example, the values for the technical factors were T1 (0), T2 (5), T3 (3), T4 (1), T5 (1), T6 (2), T7 (4), T8 (0), T9 (2), T10 (0), T11 (0), T12 (0), T13 (0), giving a TFactor of 15. Plugging this value into the TCF equation of the Use Case Point Worksheet gives a value of 0.75 for the TCF of the appointment system.

The values for the environmental factors were E1 (4), E2 (4), E3 (4), E4 (5), E5 (5), E6 (5), E7 (0), and E8 (3), giving a EFactor of 26.5. Using the EF equation of the Use Case Point Worksheet produces a value of 0.605 for the EF of the appointment system. Plugging

the TCF and EF values, along with the UUCP value computed earlier, into the Adjusted Use Case Points equation of the Worksheet computes a value of 37.2075 *Adjusted Use Case Points (UCP)*.

Now that we know the estimated size of the system by means of the value of the adjusted use case points, we are ready to estimate the *effort* required to build the system. In Karner's original work, he suggested simply multiplying the number of use case points by 20 to estimate the number of person hours required to build the system. However, based on additional experiences using use case points, a decision rule to determine the value of the *person hours multiplier (PHM)* has been created that suggests using either 20 or 28, based on the values assigned to the individual environmental factors. The decision rule is:

```
If sum of (Number of Efactors E1 through E6 Assigned Value < 3) and
    (Number of Efactors E7 and E8 Assigned Value > 3)
    <= 2
    PHM = 20
Else—If sum of (Number of Efactors E1 through E6 Assigned Value < 3) and
    (Number of Efactors E7 and E8 Assigned Value > 3)
    = 3 or 4
    PHM = 28
Else
    Rethink project; it has too high of a risk for failure
```

Based on these rules, the appointment system should use a PHM of 20. This gives an estimated number of person hours of effort of 744.15 hours (20×37.2075).

The primary advantages of using use case points over traditional estimation techniques are that they are based in object-oriented systems development and use cases, rather than traditional approaches to systems development. However, there is a risk of using use case points in that there has not been as much history using them in comparison to the traditional approaches described in Chapter 4. As such, the suggested classifications used for simple, average, and complex actors and use cases, the weighting factors for simple, average, and complex actors and use cases, and the weightings associated with the computation of the technical complexity and environmental factors may need to be modified in the future. However, at this point the suggested values seem to be the best available.

APPLYING THE CONCEPTS AT CD SELECTIONS

The basic functional and nonfunctional requirements for the CD Selections Internet sales system were developed in the previous chapters. At this point, you should go back and carefully review these requirements (see Figures 3-2, 3-12, 4-24, and 5-13).

YOUR TURN

6-4 Estimating Using Use Case Points

Consider the use case you created in Your Turn 6-2. size and effort for your use case. Using the worksheet in Figure 6-11, estimate the project

Business Process Modeling with Activity Diagrams

Based on the functional requirements identified for the Internet sales system, Alec and his team decided that there were at least four high-level activities that needed to be addressed: Place CD Orders, Maintain CD Information, Maintain CD Marketing Information, and Fill Mail Orders. As a first step toward developing a functional model of the functional requirements, Alec decided to model the overall flow of execution of the business processes as an activity diagram. Upon close review of the Place CD Orders requirements, the team identified a decision and two additional activities: Place InStore Hold and Place Special Order. These activities were based on the requirements laid out in point 3.5 of Figure 5-13. Furthermore, the team noticed that there seemed to be three separate, concurrent paths of execution implied in the requirements. The first path dealt with orders, the second addressed the maintenance of marketing information, and the third focused on maintaining the information about the CDs. Based on these new insights, Alec and his team drew the activity diagram for the Internet sales system shown in Figure 6-12.

Identify the Major Use Cases

The first four steps in writing use cases deal with reviewing the activity diagram, finding the subject boundaries, listing the primary actors and their goals, and identifying and writing overviews of the major use cases based on these results. These use cases will form the basis of the use cases that are contained on the use case diagram. Before you continue reading, take a minute and review the requirements identified in Figure 5-13 and the activity diagram we just completed (see Figure 6-12) to identify the three to nine major use cases.

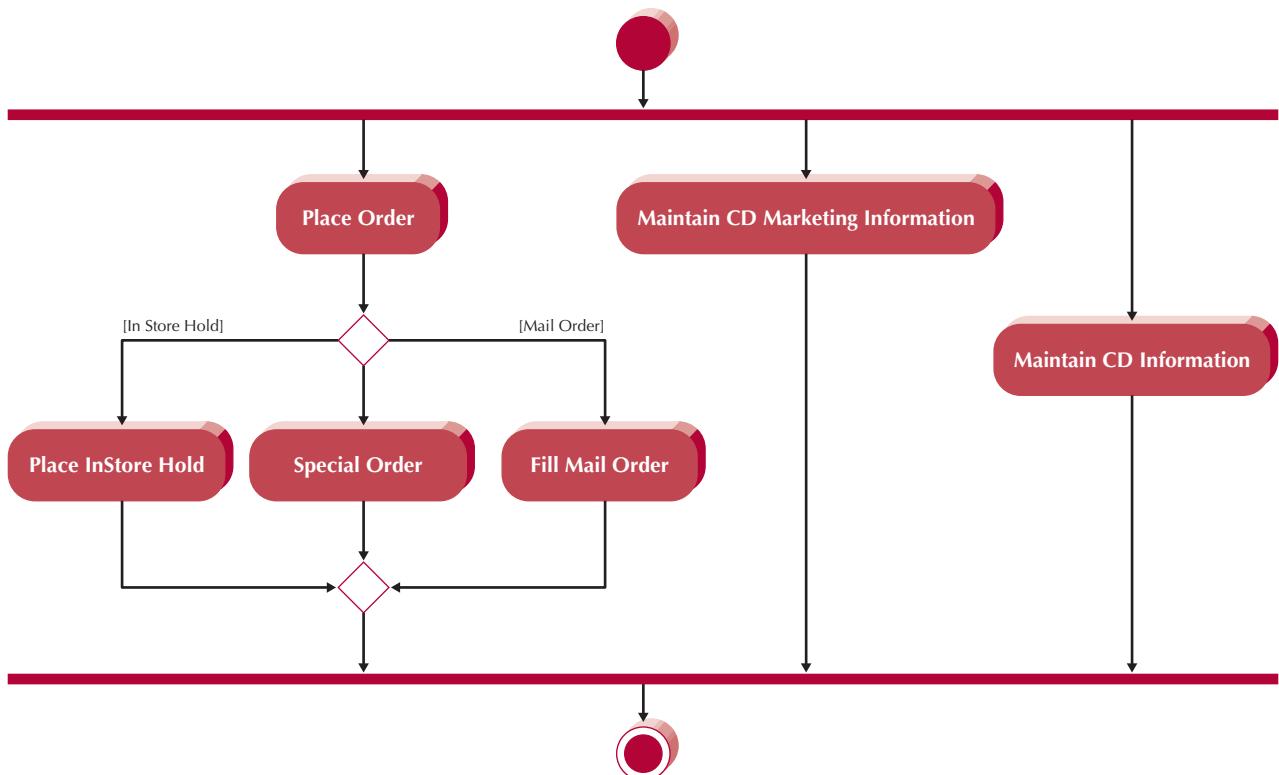


FIGURE 6-12 Activity Diagram for the CD Selections To-Be Internet Sales System

To begin with, it seems that the subject boundary should be drawn in such a manner that anything that is not part of CD Selections' Internet sales system, such as the vendors and customers, should be identified as primary actors. Therefore, these are considered outside of the scope of the system. The other potential actors identified could be the distribution system, Electronic Marketing (EM) manager, and the current CD Selections stores.

Upon closer review of Figure 5-13, it seems that the distribution system and the current CD Selections stores should be outside the scope of the Internet sales system. As such, they also should be identified as primary actors. In the case of the EM manager, it seems that the EM manager should be considered as part of the Internet sales system and therefore should not be identified as a primary actor. Remember, primary actors are only those that can be viewed as being outside of the scope of the system. The decision on whether the EM manager, the current CD Selections stores, or the distribution system is inside or outside of the system is somewhat arbitrary. From a customer's perspective, the distribution system and the current CD Selections stores could be seen as being inside of the overall system and it could be argued that the EM manager is a primary user of the Internet sales system.

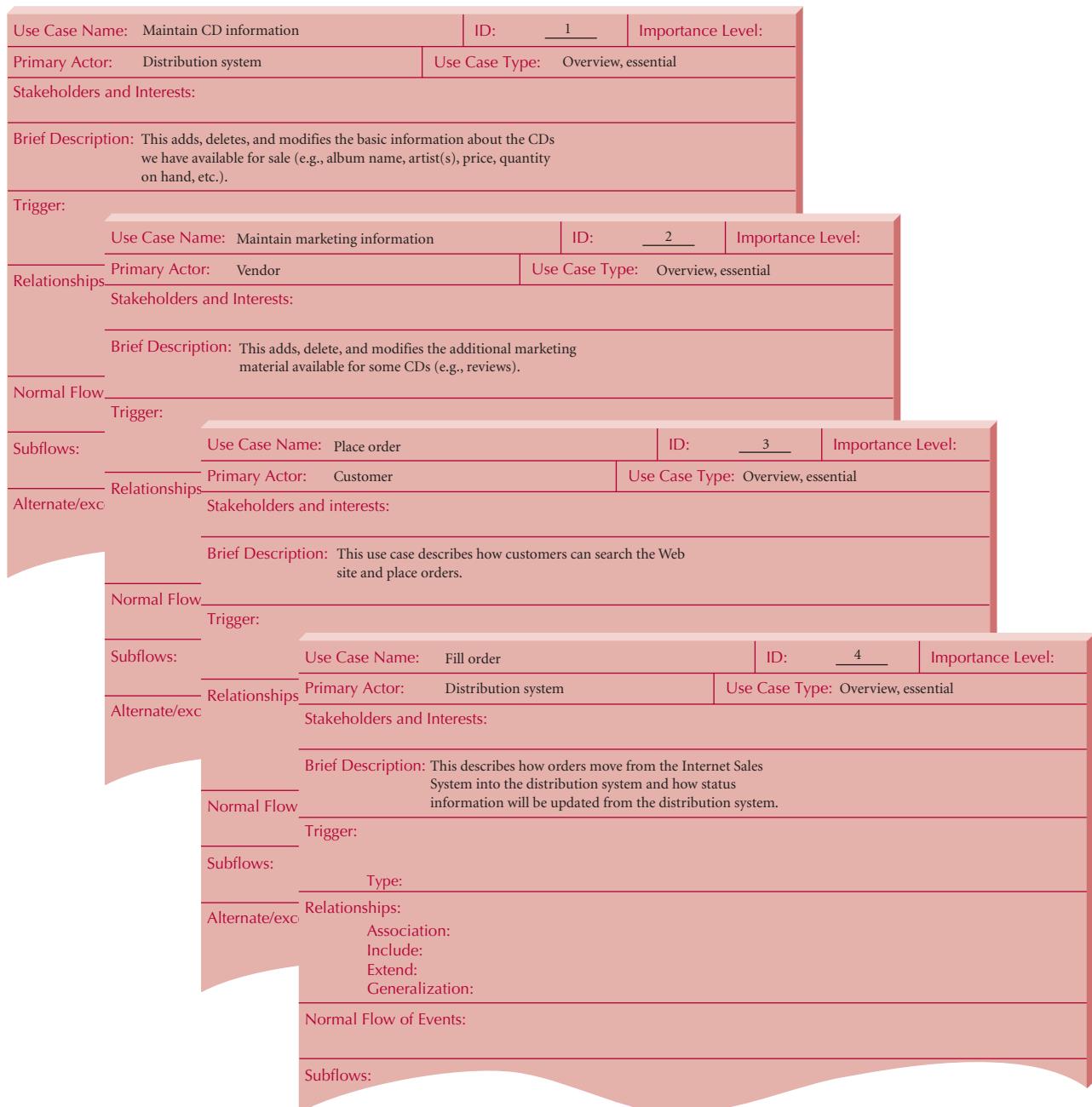
At this point in the process, it is important to make a decision and move on. During the process of writing use cases, there are ample opportunities to revisit this decision to determine whether the set of use cases identified are necessary and sufficient to describe the requirements of the Internet sales system for CD Selections. As you can see, finding the systems boundaries and listing the primary actors are heavily intertwined.

Based on the functional requirements and the activity diagram, Alec and his team identified four overview major use cases: Maintain CD Information, Maintain CD Marketing Information, Place Order, and Fill Order. You might have considered adding new CDs to the database, deleting old ones, and changing information about CDs as three separate use cases, which in fact they are. If you think about it, you should see that in addition to these three, we also need to have use cases for finding CD information and printing reports about CDs. However, our goal at this point is to develop a small set of essential use cases for the Internet sales system.

You should see the same pattern for the marketing materials. We have the same processes for recording new marketing material for a CD, changing it, deleting it, finding it, and printing it. These five activities (creating, changing, deleting, finding, and printing) are the *standard processes* usually required anytime that you have information to be stored in a database.

The project team at CD Selections identified these same four major use cases. At this point in the process, the project team began writing the overview use cases for these four. Remember, that an overview use case only has five pieces of information: use case name, ID number, primary actor, type, and a brief description. We have already identified the primary actors and have associated them with the four use cases. Furthermore, since we are just beginning the development process, all four use cases type will be Overview and Essential. Since the ID numbers are simply used for identification purposes (i.e., they act as a key in a database), their values can be assigned in a sequential manner. This only leaves us with two pieces of information for each use case to write. The use case name should be an action verb/noun phrase combination (e.g., Make Appointment). In the CD Selections Internet sales situation, Maintain CD Information, Maintain CD Marketing Information, Place Order, and Fill Order seem to capture the essence of each of the use cases. Finally, a brief description is written to describe the purpose of the use case or the goal of the primary actor using the use case. This description can range from a sentence to a short essay. The idea is to capture the primary issues in the use case and make them explicit. (See Figure 6-13.)

The fifth step is to carefully review the current set of use cases. Take a moment to review the use cases and make sure you understand them. Based on the descriptions of all four use cases, there seems to be an obvious missing use case: Maintain Order. Since the

**FIGURE 6-13** Overview Major Use Cases for CD Selections

project team did not include language in the brief description of the Place Order use case, it seems that the project team believes that maintaining an order is sufficiently different from placing an order that it should have its own use case to describe it. Furthermore, it does not seem reasonable that a customer would see placing an order and maintaining an order as the same use case. This type of process goes on until the project team feels that they have identified the major use cases for the Internet sales system.

Expanding the Major Use Cases

The sixth step is to choose one of the major use cases to expand. To help the project team to make this choice, they identified the trigger and the stakeholders and their interests for each of the major use cases.

The first use case, maintain CD information, was triggered by the distribution system distributing new information for use in the CD database. Besides the distribution system, another stakeholder was identified: EM manager. The second use case, maintain marketing materials, has a similar structure. The receipt of marketing materials received from vendors triggers it. And again, it seemed to the project team that the EM manager was an interested stakeholder. The third use case, place order, is more interesting. The trigger is the customer's actions. Again, the EM manager is an interested stakeholder. This use case has many more inputs. The fourth use case, fill order, is based on the decision logic identified in the activity diagram. However, upon further reflection, the team decided to replace this use case with three separate use cases. One for each approach to fill an order: Place Special Order, Place InStore Hold, and Fill Mail Order. The first two are controlled by actions of the customer. However, the Fill Mail Order use case has a temporal trigger: every hour the Internet sales system downloads orders into the distribution system. The final use case, maintain order, is triggered by the customer's action. However, on close review, it seems that it also has much in common with the other maintenance related use cases: maintain CD information and maintain marketing information. And like all of the other use cases, the EM manager has an interest.

Based on their review of the major use cases, the project team decided that the Place Order use case was the most interesting. The next step, number seven, is to start filling in the details of the chosen use case. At this point in time, the project team had the information to fill in the stakeholders and interests, the trigger, and the Association relationship. Remember that the association relationship is simply the communication link between the primary actor and the use case. In this use case, the Association relationship is with the Customer.

The project team then needed to gather the information needed to define the place order use case in more detail. Specifically, they needed to begin writing the Normal Flow of Events, that is, they should perform the eighth step for writing effective use cases (see Figure 6-10). This was done based on the results of the earlier analyses described in Chapter 5, as well as through a series of JAD meetings with the project sponsor and the key marketing department managers and staff who would ultimately operate the system.

The goal at this point is to describe how the use case operates. In this example, we will focus on the most complex and interesting use case, place order. The best way to begin to understand how the customer works through this use case is to visualize yourself placing a CD order over the Web, and to think about how other electronic commerce Web sites work—that is, role play. The techniques of visualizing yourself interacting with the system and of thinking about how other systems work (informal benchmarking) are important techniques that help analysts and users understand how systems work and how to write the use case. Both techniques (visualization and informal benchmarking) are commonly used in practice. It is important to remember that at this point in the development of the use case, we are only interested in the typical successful execution of the use case. If you try to think of all of the possible combinations of activities that could go on, you will never get anything written down.

In this situation, after you connect to the Web site, you probably begin searching, perhaps for a specific CD, perhaps for a category of music, but in any event, you enter some information for your search. The Web site presents a list of CDs matching your request, along with some basic information about the CDs (e.g., artist, title, and price). If one of the CDs is of interest to you, you might seek more information about it, such as the list of songs, liner notes, or reviews. Once you, the customer, find a CD you like, you will add it to your

order and perhaps continue looking for more CDs. When you are done—perhaps immediately—you will “check out” by presenting your order with information on the CDs you want and giving additional information such as mailing address and credit card number.

When you write the use case, be sure to remember the seven guidelines described earlier. One might argue that the first step is to present the customer with the home page or a form to fill in to search for an album. This is correct, but this type of step is usually very “small” compared to other steps that follow.²⁵ It is analogous to making the first step “hand the user a piece of paper.” At this point, we are only looking for the three to seven *major* steps.

The first major step performed by the system is to respond to the customer’s search inquiry, which might include a search for a specific album name or albums by a specific artist. Or, it might be the customer wanting to see all the classical or alternative CDs in stock. Or, it might be a request to see the list of special deals or CDs on “sale.” In any event, the system finds all the CDs matching the request, and shows a list of CDs in response. The user will view this response, and perhaps will decide to seek more information about one or more CDs. He or she will click on it, and the system will provide additional information. Perhaps the user will also want to see any extra marketing material that is available. The user will then select one or more CDs for purchase, decide how to take delivery of each CD, and perhaps continue with a new search. These steps correspond to events 1 through 7 in Figure 6-14.

The user may later make changes to the CDs selected, either by dropping some or changing the number ordered. This seems to be similar to an already identified separate use case: maintain order. As such, we will simple reuse that use case to handle those types of details. At some point the user will “check-out” by verifying the CDs he or she has selected for purchase, and providing information about him or herself (e.g., name, mailing address, credit card). The system will calculate the total payment and verify the credit card information with the credit card center. At this point in the transaction, the system will send an order confirmation to the customer, and the customer typically leaves the Web site. Figure 6-14 shows the use case at this point. Note that the Normal Flow of Events has been added to the form, but nothing else has changed.

The ninth step for writing uses cases (see Figure 6-10) is basically to try and simplify the current normal flow of events. Currently, we have 12 events, which is a little high. Therefore, we need to see if there are any steps that we can merge, delete, rearrange, and if there are any that we have left out. Based on this type of review, we have decided to create a separate use case that can handle the “checkout” process (events 9, 10, and 11). We also see that events 5 and 6 could be viewed as a part of the “maintain order” use case. As such, we delete events 5 and 6 and move event 8 into its place. Now we have eight events. This seems to be a reasonable number for this particular use case.

The next two steps in writing a use case addresses the handling of alternate or exceptional flows. If you remember, the Normal Flow of Events only captures the typical set of events that end in a successful transaction. In this case, CD Selections has defined success as a new order being placed. In our current use case, the project team has identified two sets of events that are exceptions to the normal flow. First, event 3 assumes that the list of recommended CDs are acceptable to the customer. However, as one of the team members pointed out, that is an unrealistic assumption. As such, two exceptional flows have been identified and written (3a-1 and 3a-2 in Figure 6-15) to handle this specific situation. Second, a customer may want to abort the entire order instead of going through the checkout process. In this case, exceptional flow 7a was created.²⁶

²⁵ Since it is so small, it violates the fourth principle (see Figure 6-5).

²⁶ Another approach would be to force the customer through the “maintain order” or “checkout” use cases. However, the marketing representatives on the project team were concerned with customer frustration. As such, the project team included it in the “place order” use case.

Confirming the Major Use Cases

Once all the use cases had been defined, the final step in the JAD session was to confirm that they were accurate. The project team had the users role play the use cases. A few minor problems were discovered and easily fixed. However, one major problem was discovered: how does a new customer get created? This was easily fixed by creating a new use case, Create New Customer, and adding it as an extension to the checkout use case. Figure 6-16 shows the revised use cases. Now the use case development process can actually start all over again or we can move on to drawing the use case diagram.

Use Case Name:	Place Order	ID:	3	Importance Level:	High			
Primary Actor:	Customer	Use Case type:	Detail, Essential					
Stakeholders and Interests:	Customer - wants to search Web site to purchase CD EM manager - wants to maximize customer satisfaction							
Brief Description:	This use case describes how customers can search the Web site and place orders.							
Trigger:	Customer visits Web site and places order							
Type:	External							
Relationships:	Association: Customer Include: Maintain order Extend: Generalization:							
Normal Flow of Events:	1. The Customer submits a search request to the system. 2. The System provides the Customer a list of recommended CDs. 3. The Customer chooses one of the CDs to find out additional information. 4. The System provides the Customer with basic information and reviews on the CD. 5. The Customer adds the CD to their shopping cart. 6. The Customer decides how to "fill" the order. 7. The Customer iterates over 3 through 5 until done shopping. 8. The Customer executes the Maintain Order use case. 9. The Customer logs in to check out. 10. The System validates the Customer's credit card information. 11. The System sends an order confirmation to the Customer. 12. The Customer leaves the Web site.							
Subflows:								
Alternate/exceptional Flows:								

FIGURE 6-14 Places Order Use Case after Step 8

**YOUR
TURN**

6-5 CD-Selections Internet Sales System

Complete the detailed use cases for the remaining overview use cases in Figure 6-16.

Use Case Name:	Place Order	ID:	3	Importance Level:	High			
Primary Actor:	Customer	Use Case Type:	Detail, Essential					
Stakeholders and Interests:	Customer—wants to search Web site to purchase CD. EM manager—wants to maximize customer satisfaction.							
Brief Description:	This use case describes how customers can search the Web site and place orders.							
Trigger:	Customer visits Web site and places order							
Type:	External							
Relationships:	Association: Customer Include: Checkout, Maintain Order Extend: Generalization:							
Normal Flow of Events:	1. Customer submits a search request to the system. 2. The System provides the Customer a list of recommended CDs. 3. The Customer chooses one of the CDs to find out additional information. 4. The System provides the Customer with basic information and reviews on the CD. 5. The Customer calls the Maintain Order use case. 6. The Customer iterates over 3 through 5 until done shopping. 7. The Customer executes the Checkout use case. 8. The Customer leaves the Web site.							
Subflows:								
Alternate/exceptional Flows:	3a-1. The Customer submits a new search request to the system. 3a-2. The Customer iterates over steps 2 through 3 until satisfied with search results or gives up. 7a. The Customer aborts the order.							

FIGURE 6-15 Places Order Use Case after Step 11

YOUR
TURN

6-6 Campus Housing

Create a set of use cases for the following high-level processes in a housing system run by the campus housing service. The campus housing service helps students find apartments. Apartment owners fill in information forms about the rental units they have available (e.g., location, number of bedrooms, monthly rent), which are then entered into a database. Students can search

through this database via the Web to find apartments that meet their needs (e.g., a two-bedroom apartment for \$400 or less per month within 1/2 mile of campus). They then contact the apartment owners directly to see the apartment and possibly rent it. Apartment owners call the service to delete their listing when they have rented their apartment(s).

Use Case Name:	Maintain CD Information	ID:	1	Importance Level:		
Primary Actor:	Distribution System	Use Case Type:	Detail, Essential			
Stakeholders and Interests:						
Brief Description: This adds, deletes, and modifies the basic information about the CDs we have available for sale (e.g., album name, artist(s), price, quantity on hand, etc.).						
Trigger: Downloads from the Distribution System						
Type: External						
Relationships:						
Association: Distribution System						
Include:						
Use Case Name:	Maintain Marketing Information	ID:	2	Importance Level:		
Primary Actor:	Vendor	Use Case Type:	Detail, Essential			
Stakeholders and Interests: Vendor—wants to ensure marketing information is as current as possible. EM Manager—wants marketing information to be correct to maximize sales.						
Brief Description: This adds, deletes, and modifies the marketing material available for some CDs (e.g., reviews).						
Trigger: Materials arrive from vendors, distributors, wholesalers, record companies, and articles from trade magazines						
Type: External						
Relationships:						
Association: Vendor						
Include:						
Use Case Name:	Place Order	ID:	3	Importance Level:		
Primary Actor:	Customer	Use Case Type:	Detail, Essential			
Stakeholders and Interests: Customer—wants to search Web site to purchase CD. EM Manager—wants to maximize Customer satisfaction.						
Brief Description: This use case describes how customers can search the Web site and place orders.						
Trigger: Customer visits Web site and places order.						
Type: External						
Relationships:						
Association: Customer						
Include: Checkout, Maintain Order						
Extend:						
Generalization :						

FIGURE 6-16A Revised Major Use Cases for CD Selections (Continues)

Use Case Name:	Maintain Order	ID:	5	Importance Level:	High									
Primary Actor:	Customer	Use Case Type:		Detail, Essential										
Stakeholders and Interests:	Customer—wants to modify order. EM Manager—wants to ensure high customer satisfaction.													
Brief Description: This use case describes how a Customer can cancel or modify an open or existing order.														
Trigger: Customer visits Web site and requests to modify a current order.														
Type:	External													
Relationships:														
Association:														
Include:														
Use Case Name:	Checkout	ID:	6	Importance Level:	High									
Primary Actor:	Customer	Use Case Type:		Detail, Essential										
Stakeholders and Interests:	Customer—wants to finalize the order. Credit Card Center—wants to provide effective and efficient service to CD selections. EM Manager—wants to maximize order closings.													
Brief Description: This use case describes how the customer completes an order including credit card authorization.														
Trigger: Customer signals the system they want to finalize their order.														
Type:	External													
Relationships:														
Association:	Credit Card Center													
Include:	Maintain Order													
Use Case Name:	Create New Customer	ID:	7	Importance Level:	High									
Primary Actor:	Customer	Use Case Type:		Detail, Essential										
Stakeholders and Interests:	Customer—wants to be able to purchase CDs from CD selections. EM Manager—wants to increase CD selections Customer base.													
Brief Description: This use case describes how a customer is added to the Customer database.														
Trigger: An unknown Customer attempts to checkout.														
Type:	External													
Relationships:														
Association:														
Include:														
Extend:	Customer													
Generalization :														

FIGURE 6-16B Revised Major Use Cases for CD Selections (Continues)

Use Case Name: Place Special Order		ID: 8	Importance Level: High
Primary Actor: Customer		Use Case Type: Detail, Essential	
Stakeholders and Interests: Customer—wants to be able to place a special order of CDs for in store pick up. EM manager—wants to increase sales associated with the Internet Sales system. Bricks and Mortar Store Manager—wants to increase sales associated with the store.			
Brief Description: This use case describes how a Customer places a special order using the Internet Sales system.			
Trigger: Customer selects CD on order for a special order at bricks and mortar store.			
Type: External			
Relationships: Association: Bricks and mortar store Include:			
Use Case Name: Place InStore Hold		ID: 9	Importance Level: High
Primary Actor: Customer		Use Case Type: Detail, Essential	
Stakeholders and Interests: Customer—wants to be able to place an in store hold a CD for In Store pick up. EM manager—wants to increase sales associated with the Internet Sales system. Bricks and Mortar Store Manager—wants to increase sales associated with the store.			
Brief Description: This use case describes how a Customer places an in store hold using the Internet Sales system.			
Trigger: Customer selects CD on order for an in store hold to be picked up at bricks and mortar store.			
Type: External			
Relationships: Association: Bricks and mortar store Include:			
Use Case Name: Fill Mail Order		ID: 10	Importance Level: High
Primary Actor: Customer		Use Case Type: Detail, Essential	
Stakeholders and Interests: Mail Order Distribution System—wants to complete order processing in a timely manner. Customer—wants to receive order in a timely manner. EM Manager—wants to maximize order throughput.			
Brief Description: This describes how mail orders move from the Internet Sales system into the distribution system and how status information will be updated from the distribution system.			
Trigger: Every hour the Distribution System will initiate a trading of information with the Internet Sales system.			
Type: Temporal			
Relationships: Association: Distribution System Include: Extend: Maintain Order Generalization :			

FIGURE 6-16C Revised Major Use Cases for CD Selections (Continued)

Creating the Use Case Diagram

Creating a use case diagram from the detailed use case descriptions is very easy. Since a use case diagram only shows the subject boundary, the use cases themselves, the actors, and the various associations between these components, they support information hiding. Use case diagrams only provide a pictorial overview of the detailed use cases. There are four major steps to drawing a use case diagram.

Draw the Subject Boundary In this case, place a box on the use case diagram to represent the Internet sales system, and place the system's name either inside or on top of the box. Based on a review of Figure 6-16, be sure to draw the box large enough to contain the use cases. Also, be sure to leave space on the outside of the box for the actors.

Place the Use Cases on the Diagram The next step is to add the use cases. Place the number of use cases inside the system boundary to correspond with the number of functions that the system needs to perform. There should be no more than three to nine use cases on the model. Can you identify the use cases that need to be placed in the system boundary by examining Figure 6-16?

Identify the Actors Once the use cases are placed on the diagram, you will need to place the actors on it. Normally, it is fairly easy to place the actors near their respective use cases. Look at the detailed use cases in Figure 6-16 and see if you can identify the five "actors" that belong on the use case diagram.

Hopefully you have listed the distribution system, the existing bricks-and-mortar stores, customer, vendor, and credit card center. At this point, there are no specialized actors that need to be included.

Add Associations The last step is to draw lines connecting the actors with the use cases with which they interact. See Figure 6-17 for the use case diagram that we have created.

YOUR TURN

6-7 Identifying Generalization Associations in Use Cases and Specialized Actors

The use case diagram for the Internet sales systems does not include any generalization relationships. See if you can come up with one example for a use case and another example for an actor that may be helpful for CD

Selections to add to the use case diagram shown in Figure 6-17. Describe how the development effort may benefit from including your examples.

YOUR TURN

6-8 Drawing a Use Case Diagram

Previously in Your Turn 6-6, you identified use cases for a campus housing service that helps students find

apartments. Based on those use cases, create a use case diagram.

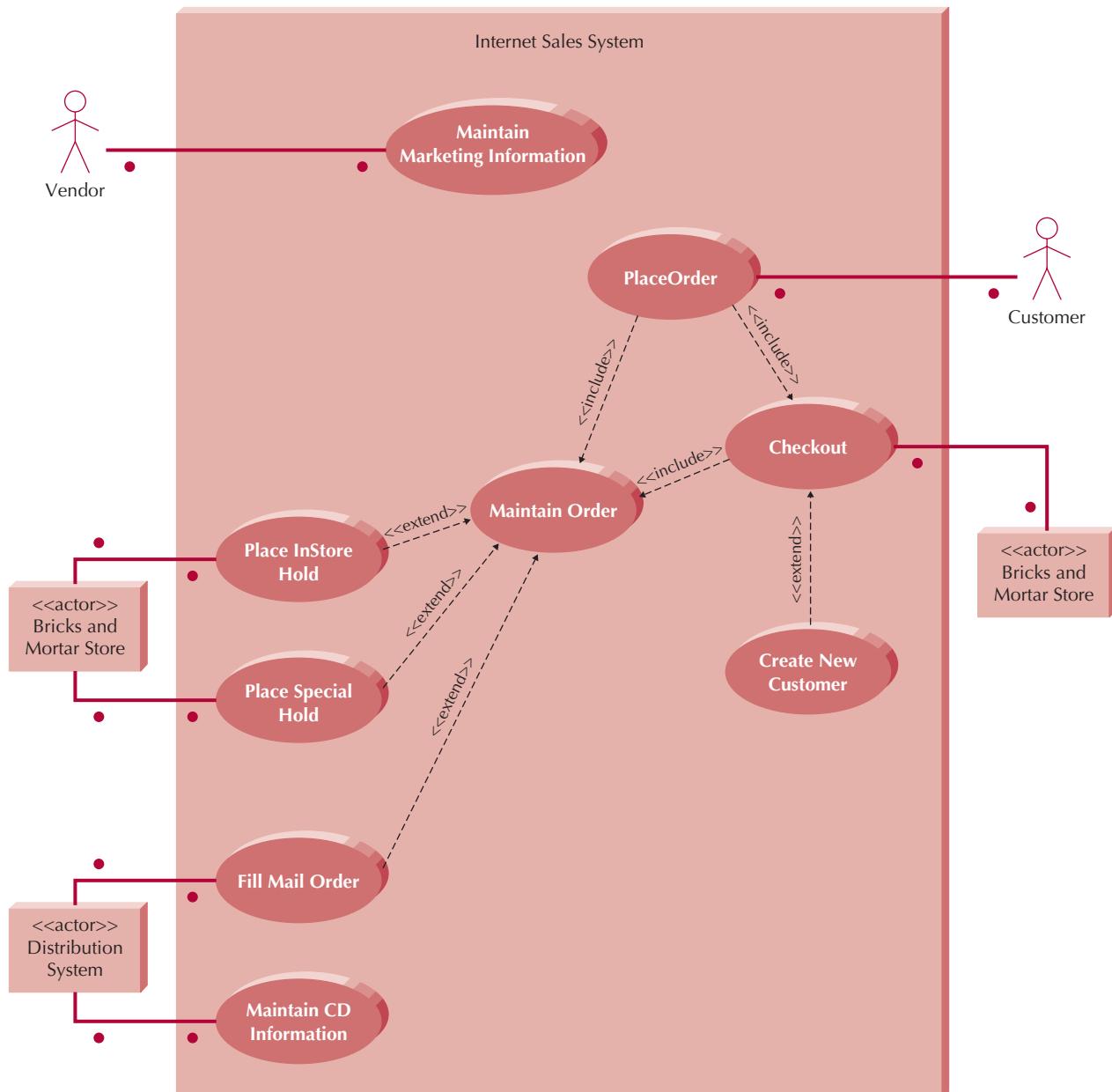


FIGURE 6-17 Use Case Diagram for the CD Selections Internet Sales System

Refine Project Size and Effort Estimation Using Use Case Points

Now that the functional modeling aspect of the system development effort has been completed, Alec and his team decide that they need to refine the estimation of the size and effort to build the system. Based on the completed use cases (see Figure 6-16) and the use case diagram (see Figure 6-17), Alec could now use the use cases as the basis for the estimation using use case points.

First, Alec had to classify each actor and use case as being simple, average, or complex. In the case of the actors, Bricks and Mortar store and Distribution System had a well-defined

API. As such they were classified as simple actors. The Credit Card Center was considered to be average, while the Vendor and Customer actors were classified as being complex. This gave an Unadjusted Actor Weight Total Value of 10. (See Figure 6-18.)

Second, Alec classified each use case based on the number of unique transactions that each had to handle. In this case, there were three simple use cases (Place InStore Hold, Place Special Order, and Fill Mail Order), one average use case (Create New Customer), and five complex use cases (Place Order, Checkout, Maintain Order, Maintain CD Information, and Maintain CD Marketing Information). This assigns a value of 100 to the Unadjusted Use Case Weight Total (see Figure 6-18).

Third, Alec computed a value of 110 for the Unadjusted Use Case Points. Fourth, he rated each of the technical complexity factors, rated each of the environmental factors, and computed the values for TCF and EF (see Figure 6-18). Fifth, using the Unadjusted Use Case Points and the TCF and EF values, Alec calculated a value of 134.992 for Adjusted Use Case Points. Sixth, based on the decision rule to determine whether to use 20 or 28 as the value of the person hours multiplier, Alec realized that he should use 28. Finally, Alec was able to estimate the remaining effort for the project to be 3,779.776 hours. Based on the new effort estimate, Alec updated the schedule.

SUMMARY

Business Process Modeling with Activity Diagrams

Even from an object-oriented systems development point of view, business process modeling has been shown to be valuable. The one major hazard of business process modeling is that it focuses the systems development process in a functional decomposition direction. However, if used carefully, it can enhance the development of object-oriented systems. UML supports process modeling using an activity diagram. Activity diagrams are comprised of activities or actions, objects, control flows, object flows, and a set of seven different control nodes (initial, final-activity, final-flow, decision, merge, fork, and join). Furthermore, swimlanes can be used to enhance the readability of the diagrams. The activity diagram is very useful in aiding the analyst in identifying the relevant use cases for the information system being developed.

Use Case Descriptions

Use cases are the primary method of documenting requirements for object-oriented systems. They represent a functional view of the system being developed. There are overview and detail use cases. Overview use cases comprise the use case name, ID number, primary actor, type, and a brief description. Detailed use cases extend the overview use case with the identification and description of the stakeholders and their interest, the trigger and its type, the relationships in which the use case participates (association, extend, generalization, and include), the normal flow of events, the subflows, and any alternate or exception flows to the normal flow of events. There are seven guidelines (see Figure 6-5) and thirteen steps (see Figure 6-10) to writing effective use cases. The thirteen steps can be summed up into three aggregate groups of steps: identify the major use cases (steps 1 to 5), expand the major use cases (steps 6 to 11), and confirm the major use cases (steps 12 to 13).

Use Case Diagrams

Use case diagrams are simply a graphical overview of the set of use cases contained in the system. They illustrate the main functionality of a system and the actors that interact with

Unadjusted Actor Weighting Table:					
Actor Type	Description	Weighting Factor	Number	Result	
Simple	External System with well-defined API	1	2	2	
Average	External System using a protocol-based interface, e.g., HTTP, TCP/IP, or a database	2	1	2	
Complex	Human	3	2	6	
Unadjusted Actor Weight Total (UAW)				10	
Unadjusted Use Case Weighting Table:					
Use Case Type	Description	Weighting Factor	Number	Result	
Simple	1–3 transactions	5	3	15	
Average	4–7 transactions	10	1	10	
Complex	>7 transactions	15	5	75	
Unadjusted Use Case Weight Total (UUCW)				100	
Unadjusted Use Case Points (UUCP) = UAW + UUCW 110 = 10 + 100					
Technical Complexity Factors:					
Factor Number	Description	Weight	Assigned Value (0–5)	Weighted Value	Notes
T1	Distributed system	2.0	5	10.0	
T2	Response time or throughput performance objectives	1.0	5	5.0	
T3	End-user online efficiency	1.0	5	5.0	
T4	Complex internal processing	1.0	4	4.0	
T5	Reusability of code	1.0	3	3.0	
T6	Easy to install	0.5	3	1.5	
T7	Ease of use	0.5	5	2.5	
T8	Portability	2.0	4	8.0	
T9	Ease of change	1.0	3	3.0	
T10	Concurrency	1.0	3	3.0	
T11	Special security objectives included	1.0	5	5.0	
T12	Direct access for third parties	1.0	5	5.0	
T13	Special User training required	1.0	3	3.0	
Technical Factor Value (TFactor)				58.0	
Technical Complexity Factor (TCF) = 0.6 + (0.01 × TFactor) 1.18 = 0.6 + (0.01 × 58)					
Environmental Factors:					
Factor Number	Description	Weight	Assigned Value (0 – 5)	Weighted Value	Notes
E1	Familiarity with system development process being used	1.5	1	1.5	
E2	Application experience	0.5	2	1.0	
E3	Object-oriented experience	1.0	0	0.0	
E4	Lead analyst capability	0.5	3	1.5	
E5	Motivation	1.0	4	4.0	
E6	Requirements stability	2.0	4	8.0	
E7	Part time staff	-1.0	0	0.0	
E8	Difficulty of programming language	-1.0	4	-4.0	
Environmental Factor Value (EFactor)				12.0	
Environmental Factor (EF) = 1.4 + (-0.03 * EFactor) 1.04 = 1.4 + (-0.03 × 12)					
Adjusted Use Case Points (UCP) = UUCP * TCF * ECF 134.992 = 110 × 1.18 × 1.04					
Person Hours Multiplier (PHM) PHM = 28					
Person Hours = UCP × PHM 3,779.776 = 134.992 × 28					

FIGURE 6-18 Use Case Points Estimation for the Internet Sales Systems

the system. The diagram includes actors, which are people or things that derive value from the system, and use cases that represent the functionality of the system. The actors and use cases are separated by a subject boundary and connected by lines representing associations. At times, actors are specialized versions of more general actors. Similarly, use cases can extend or include other use cases. Creating use-case diagrams is a four-step process (see Figure 6-10) whereby the analyst draws the subject boundary, adds the use cases to the diagram, identifies the actors, and finally adds appropriate associations to connect use cases and actors together. This process is simple because the written description of the use cases is created first.

Refine Project Size and Effort Estimation with Use Case Points

Use case points are a relatively new size and effort estimation technique that is based on ideas similar to function point analysis. Use case points are founded on the two primary constructs associated with use case analysis: actors and use cases. Like function points, use case points have a set of factors used to modify their raw value: technical complexity factors and environmental factors. Technical complexity factors address the complexity of the project under consideration while the environmental factors deal with the level of experience of the development staff. Based on the number of use case points, the estimated effort required can be computed.

KEY TERMS

Action	Final-flow node	Scenario
Activity	Flow of events	Simple actors
Activity diagram	Fork node	Simple use cases
Actor	Generalization relationship	Specialized actor
Adjusted use case points (UCP)	Guard condition	Stakeholders
Alternate flows	Importance level	Standard processes
Application program interface (API)	Include relationship	Subflows
Association relationship	Inherit	Subject boundary
Average actors	Inheritance	SVDPI
Average use cases	Initial node	Swimlanes
Black-hole activities	Iterate	Technical complexity factor (TCF)
Brief description	Join node	Technical factor value (TFactor)
Complex actors	Logical model	Temporal trigger
Complex use cases	Merge node	Trigger
Control flow	Miracle activity	Unadjusted actor weight total (UAW)
Control node	Object flow	Unadjusted use case points (UUCP)
Decision node	Object node	Unadjusted use case weight total (UUCW)
Detail use case	Overview use case	Use case
Effort	Normal flow of events	Use-case description
Environmental factor (EF)	Packages	Use-case diagram
Environmental factor value (EFactor)	Person hours multiplier (PHM)	Use-case ID number
Essential use case	Physical model	Use-case name
Estimation	Primary actor	Use-case points
Exceptional flows	Process models	Use-case type
Extend relationship	Real use case	
External trigger	Relationships	
Final-activity node	Role	

QUESTIONS

1. Why is business process modeling important?
2. What is the purpose of an activity diagram?
3. What is the difference between an activity and an action?
4. What is the purpose of a fork node?
5. What are the different types of control nodes?
6. What is the difference between a control flow and an object flow?
7. What is an object node?
8. How is use case diagramming related to functional modeling?
9. Explain the following terms. Use layman's language as though you were describing them to a user: (a) actor; (b) use case; (c) subject boundary; (d) relationship.
10. Every association must be connected to at least one _____ and one _____. Why?
11. What is CRUD? Why is this useful?
12. How does a detail use case differ from an overview use case?
13. How does an essential use case differ from a real use case?
14. What are the major elements of an overview use case?
15. What are the major elements of a detail use case?
16. Describe how to create use cases.
17. Why do we strive to have about three to nine major use cases in a business process?
18. Describe how to create use case diagrams.
19. What are some heuristics for creating a use case diagram?
20. Why is iteration important in creating use cases?
21. What is the viewpoint of a use case, and why is it important?
22. What are some guidelines for designing a set of use cases? Give two examples of the extend associations on a use case diagram. Give two examples for the include association.
23. Which of the following could be an actor found on a use case diagram? Why?
 - Ms. Mary Smith
 - Supplier
 - Customer
 - Internet customer
 - Mr. John Seals
 - Data entry clerk
 - Database administrator
24. What is a use case point? What is it used for?
25. Describe the process to estimate systems development based on use cases.

EXERCISES

- A. Investigate the Web site for Rational Software (www.rational.com) and its repository of information about UML. Write a paragraph news brief on the current state of UML (e.g., the current version and when it will be released; future improvements; etc.).
- B. Investigate the Object Management Group. Write a brief memo describing what it is, its purpose, and its influence on UML and the object approach to systems development. (*Hint:* A good resource is www.omg.org.)
- C. Create an activity diagram and a set of detail use case descriptions for the process of buying glasses from the viewpoint of the patient, but do not bother to identify the flow of events within each use case. The first step is to see an eye doctor who will give you a prescription. Once you have a prescription, you go to a glasses store, where you select your frames and place the order for your glasses. Once the glasses have been made, you return to the store for a fitting and pay for the glasses.
- D. Draw a use case diagram for the process of buying glasses in Exercise C.
- E. Create an activity diagram and a set of detail use case descriptions for the following dentist office system,

but do not bother to identify the flow of events within each use case. Whenever new patients are seen for the first time, they complete a patient information form that asks their name, address, phone number and brief medical history, which are stored in the patient information file. When a patient calls to schedule a new appointment or change an existing appointment, the receptionist checks the appointment file for an available time. Once a good time is found for the patient, the appointment is scheduled. If the patient is a new patient, an incomplete entry is made in the patient file; the full information will be collected when they arrive for their appointment. Because appointments are often made so far in advance, the receptionist usually mails a reminder postcard to each patient two weeks before their appointment.

- F. Draw a use case diagram for the dentist office system in Exercise E.
- G. Complete the detail use case descriptions for the dentist office system in Exercise E by identifying the normal flow of events, subflows, and alternate/exceptional flows within the use cases.

- H.** Create an activity diagram and a set of detail use case descriptions for an online university registration system. The system should enable the staff of each academic department to examine the courses offered by their department, add and remove courses, and change the information about them (e.g., the maximum number of students permitted). It should permit students to examine currently available courses, add and drop courses to and from their schedules, and examine the courses for which they are enrolled. Department staff should be able to print a variety of reports about the courses and the students enrolled in them. The system should ensure that no student takes too many courses and that students who have any unpaid fees are not permitted to register.
- I.** Draw a use case diagram for the online university registration system in Exercise H.
- J.** Create an activity diagram and a set of detail use case descriptions for the following system. A Real Estate Inc. (AREI) sells houses. People who want to sell their houses sign a contract with AREI and provide information on their house. This information is kept in a database by AREI and a subset of this information is sent to the city-wide multiple listing service used by all real estate agents. AREI works with two types of potential buyers. Some buyers have an interest in one specific house. In this case, AREI prints information from its database, which the real estate agent uses to help show the house to the buyer (a process beyond the scope of the system to be modeled). Other buyers seek AREI's advice in finding a house that meets their needs. In this case, the buyer completes a buyer information form that is entered into a buyer data base, and AREI real estate agents use its information to search AREI's data base and the multiple listing service for houses that meet their needs. The results of these searches are printed and used to help the real estate agent show houses to the buyer.
- K.** Draw a use case diagram for the real estate system in Exercise J.
- L.** Create an activity diagram and a set of detail use case descriptions for the following system. A Video Store (AVS) runs a series of fairly standard video stores. Before a video can be put on the shelf, it must be cataloged and entered into the video database. Every customer must have a valid AVS customer card in order to rent a video. Customers rent videos for three days at a time. Every time a customer rents a video, the system must ensure that they do not have any overdue videos. If so, the overdue videos must be returned and an overdue fee paid before customer can rent more videos. Likewise, if the customer has returned overdue videos, but has not paid the overdue fee, the fee must be paid before new videos can be rented. Every morning, the store manager prints a report that lists overdue videos. If a video is two or more days overdue, the manager calls the customer to remind them to return the video. If a video is returned in damaged condition, the manager removes it from the video database and may sometimes charge the customer.
- M.** Draw a use case diagram for the video system in Exercise L.
- N.** Create an activity diagram and a set of detail use case descriptions for a health club membership system. When members join the health club, they pay a fee for a certain length of time. Most memberships are for one year, but memberships as short as two months are available. Throughout the year, the health club offers a variety of discounts on their regular membership prices (e.g., two memberships for the price of one for Valentine's Day). It is common for members to pay different amounts for the same length of membership. The club wants to mail out reminder letters to members asking them to renew their memberships one month before their memberships expire. Some members have become angry when asked to renew at a much higher rate than their original membership contract, so the club wants to track the price paid so that the manager can override the regular prices with special prices when members are asked to renew. The system must track these new prices so that renewals can be processed accurately. One of the problems in the health club industry is the high turnover rate of members. While some members remain active for many years, about half of the members do not renew their memberships. This is a major problem, because the health club spends a lot in advertising to attract each new member. The manager wants the system to track each time a member comes into the club. The system will then identify the heavy users, and generate a report so the manager can ask them to renew their memberships early, perhaps offering them a reduced rate for early renewal. Likewise, the system should identify members who have not visited the club in more than a month, so the manager can call them and attempt to re-interest them in the club.
- O.** Draw a use case diagram for the system in Exercise N.
- P.** Create an activity diagram and a set of detail use case descriptions for the following system. Picnics R Us (PRU) is a small catering firm with five employees. During a typical summer weekend, PRU caters fifteen picnics with twenty to fifty people each. The business has grown rapidly over the past year and the owner wants to install a new computer system for managing the ordering and buying process. PUR has a set of ten standard menus. When potential customers call, the receptionist describes the menus to them. If the customer decides to book a

picnic, the receptionist records the customer information (e.g., name, address, phone number, etc.) and the information about the picnic (e.g., place, date, time, which one of the standard menus, total price) on a contract. The customer is then faxed a copy of the contract and must sign and return it along with a deposit (often a credit card or by check) before the picnic is officially booked. The remaining money is collected when the picnic is delivered. Sometimes, the customer wants something special (e.g., birthday cake). In this case, the receptionist takes the information and gives it to the owner who determines the cost; the receptionist then calls the customer back with the price information. Sometimes the customer accepts the price, other times, the customer requests some changes that have to go back to the owner for a new cost estimate. Each week, the owner looks through the picnics scheduled for that weekend and orders the supplies (e.g., plates) and food (e.g., bread, chicken) needed to make them. The owner would like to use the system for marketing as well. It should be able to track how customers learned about PUR, and identify repeat customers, so that PUR can mail special offers to them. The owner also wants to track the picnics on which PUR sent a contract, but the customer never signed the contract and actually booked a picnic.

- Q.** Draw a use case diagram for the system in Exercise P.
R. Create an activity diagram and a set of detail use case descriptions for the following system. Of-the-Month Club (OTMC) is an innovative young firm that sells memberships to people who have an interest in certain products. People pay membership fees for one year and each month receive a product by mail. For example, OTMC has a coffee-of-the-month club that sends members one pound of special coffee each month. OTMC currently has six memberships (coffee, wine, beer, cigars, flowers, and computer games) each of which costs a different amount. Customers usually belong to just one, but some belong to two or more. When people join OTMC, the telephone operator records the name, mailing address, phone number, e-mail address, credit card information, start date, and membership service(s) (e.g., coffee). Some customers request a double or triple

membership (e.g., two pounds of coffee, three cases of beer). The computer game membership operates a bit differently from the others. In this case, the member must also select the type of game (action, arcade, fantasy/science-fiction, educational, etc.) and age level. OTMC is planning to greatly expand the number of memberships it offers (e.g., video games, movies, toys, cheese, fruit, and vegetables) so the system needs to accommodate this future expansion. OTMC is also planning to offer 3-month and 6-month memberships.

- S.** Draw a use case diagram for the system in Exercise R.
T. Create an activity diagram and a set of detail use case descriptions for a university library borrowing system (do not worry about catalogue searching, etc.). The system will record the books owned by the library and will record who has borrowed what books. Before someone can borrow a book, he or she must show a valid ID card, which is checked to ensure that it is still valid against the student database maintained by the registrar's office (for student borrowers), the faculty/staff database maintained by the personnel office (for faculty/staff borrowers), or against the library's own guest database (for individuals issued a "guest" card by the library). The system must also check to ensure that the borrower does not have any overdue books or unpaid fines before he or she can borrow another book. Every Monday, the library prints and mails postcards to those people with overdue books. If a book is overdue by more than two weeks, a fine will be imposed and a librarian will telephone the borrower to remind him or her to return the book(s). Sometimes books are lost or are returned in damaged condition. The manager must then remove them from the database and will sometimes impose a fine on the borrower.
U. Draw a use case diagram for the system in Exercise T.
V. Consider the application that is used at your school to register for classes. Complete a use case point worksheet to estimate the effort to build such an application. You will need to make some assumptions about the application's interfaces and the various factors that affect its complexity.
W. For exercises G, H, J, L, N, P, R, and T, complete a use case point worksheet to estimate the effort to build such an application.

MINICASES

- Williams Specialty Company is a small printing and engraving organization. When Pat Williams, the owner, brought computers into the business office five years ago, the business was very small and very simple. Pat was able to utilize an inexpensive PC-based accounting

system to handle the basic information processing needs of the firm. As time has gone on, however, the business has grown and the work being performed has become significantly more complex. The simple accounting software still in use is no longer adequate to

keep track of many of the company's sophisticated deals and arrangements with its customers.

Pat has a staff of four people in the business office who are familiar with the intricacies of the company's record-keeping requirements. Pat recently met with her staff to discuss her plan to hire an IS consulting firm to evaluate their information system needs and recommend a strategy for upgrading their computer system. The staff is excited about the prospect of a new system, since the current system causes them much aggravation. No one on the staff has ever done anything like this before, however, and they are a little wary of the consultants who will be conducting the project.

Assume that you are a systems analyst on the consulting team assigned to the Williams Specialty Co. engagement. At your first meeting with the Williams staff, you want to be sure that they understand the work that your team will be performing, and how they will participate in that work.

- a. Explain in clear, nontechnical terms, the goals of the Analysis phase of the project.
 - b. Explain in clear, nontechnical terms, how use cases and use case diagram will be used by the project team. Explain what these models are, what they represent in the system, and how they will be used by the team.
2. Professional and Scientific Staff Management (PSSM) is a unique type of temporary staffing agency. Many organizations today hire highly skilled, technical employees on a short-term, temporary basis, to assist with special projects or to provide a needed technical skill. PSSM negotiates contracts with its client companies in which it agrees to provide temporary staff in specific job categories for a specified cost. For example, PSSM has a contract with an oil and gas exploration company in which it agrees to supply geologists with at least a master's degree for \$5,000 per week. PSSM has contracts with a wide range of companies and can place almost any type of professional or scientific staff members, from computer programmers to geologists to astrophysicists.

When a PSSM client company determines that it will need a temporary professional or scientific employee, it issues a staffing request against the contract it had previously negotiated with PSSM. When a staffing request is received by PSSM's contract manager, the contract number referenced on the staffing request is entered into the contract database. Using

information from the database, the contract manager reviews the terms and conditions of the contract and determines whether the staffing request is valid. The staffing request is valid if the contract has not expired, the type of professional or scientific employee requested is listed on the original contract, and the requested fee falls within the negotiated fee range. If the staffing request is not valid, the contract manager sends the staffing request back to the client with a letter stating why the staffing request cannot be filled, and a copy of the letter is filed. If the staffing request is valid, the contract manager enters the staffing request into the staffing request database as an outstanding staffing request. The staffing request is then sent to the PSSM placement department.

In the Placement Department, the type of staff member, experience, and qualifications requested on the staffing request are checked against the database of available professional and scientific staff. If a qualified individual is found, he or she is marked "reserved" in the staff database. If a qualified individual cannot be found in the database, or is not immediately available, the Placement Department creates a memo that explains the inability to meet the staffing request and attaches it to the staffing request. All staffing requests are then sent to the Arrangements Department.

In the Arrangement Department the prospective temporary employee is contacted and asked to agree to the placement. After the placement details have been worked out and agreed to, the staff member is marked "placed" in the staff database. A copy of the staffing request and a bill for the placement fee is sent to the client. Finally, the staffing request, the "unable to fill" memo (if any), and a copy of the placement fee bill is sent to the contract manager. If the staffing request was filled, the contract manager closes the open staffing request in the staffing request database. If the staffing request could not be filled the client is notified. The staffing request, placement fee bill, and "unable to fill" memo are then filed in the contract office.

- a. Create an activity diagram for the business process described above.
- b. Develop a use case for each major activity identified in the activity diagram.
- c. Create a use case diagram for the system described above.

CHAPTER 7

STRUCTURAL MODELING

A structural, or conceptual, model describes the structure of the data that supports the business processes in an organization. During the analysis phase, the structural model presents the logical organization of data without indicating how the data are stored, created, or manipulated so that analysts can focus on the business without being distracted by technical details. Later, during the design phase, the structural model is updated to reflect exactly how the data will be stored in databases and files. This chapter describes Class-Responsibility-Collaboration (CRC) cards, class diagrams, and object diagrams that are used to create the structural model.

OBJECTIVES

- Understand the rules and style guidelines for creating CRC cards, class diagrams, and object diagrams.
- Understand the processes used to create CRC cards, class diagrams, and object diagrams.
- Be able to create CRC cards, class diagrams, and object diagrams.
- Understand the relationship between the structural and use-case models.

CHAPTER OUTLINE

Introduction	Elements of a Class Diagram
Structural Models	Simplifying Class Diagrams
Classes, Attributes, and Operations	Object Diagrams
Relationships	Creating CRC Cards and Class Diagrams
Class-Responsibility-Collaboration Cards	Object Identification
Responsibilities and Collaborations	Building CRC Cards and Class Diagrams
Elements of a CRC Card	Applying the Concepts at CD Selections
Class Diagrams	Summary

INTRODUCTION

During the analysis phase, analysts create functional models to represent how the business system will behave. At the same time, analysts need to understand the information that is used and created by the business system (e.g., customer information, order information). In this chapter, we discuss how the data underlying the behavior modeled in the use cases are organized and presented.

A *structural model* is a formal way of representing the objects that are used and created by a business system. It illustrates people, places, or things about which information is captured, and how they are related to each other. The structural model is drawn using an iterative process in which the model becomes more detailed and less “conceptual” over time. In the analysis

phase, analysts draw a *conceptual model*, which shows the logical organization of the objects without indicating how the objects are stored, created, or manipulated. Because this model is free from any implementation or technical details, the analysts can focus more easily on matching the model to the real business requirements of the system.

In the design phase, analysts evolve the conceptual structural model into a design model that reflects how the objects will be organized in databases and files. At this point, the model is checked for redundancy and the analysts investigate ways to make the objects easy to retrieve. The specifics of the design model will be discussed in detail in the design chapters.

In this chapter, we focus on creating a conceptual structural model of the objects using CRC cards and class diagrams. Using these techniques, it is possible to show all of the objects of a business system. We first describe structural models and their elements. Then, we describe CRC cards, class diagrams, and object diagrams. Finally, we describe how to create structural models using CRC cards and class diagrams and how the structural model relates to the use cases and use case diagram that you learned about in Chapter 6.

STRUCTURAL MODELS

Every time a systems analyst encounters a new problem to solve, he or she must learn the underlying problem domain. The goal of the analyst is to discover the key objects contained in the problem domain and to build a structural model of the objects. Object-oriented modeling allows the analyst to reduce the “semantic gap” between the underlying problem domain and the evolving structural model. However, the real world and the world of software are very different. The real world tends to be messy, whereas the world of software must be neat and logical. As such, an exact mapping between the structural model and the problem domain may not be possible. In fact, it may not even be desirable.

One of the primary purposes of the structural model is to create a vocabulary that can be used by both the analyst and users. Structural models represent the things, ideas, or concepts, that is, the objects, contained in the domain of the problem. They also allow the representation of the relationships between the things, ideas, or concepts. By creating a structural model of the problem domain, the analyst creates the vocabulary necessary for the analyst and users to communicate effectively.

One important thing to remember is that, at this stage of development, the structural model does not represent software components or classes in an object-oriented programming language, even though the structural model does contain analysis classes, attributes, operations, and relationships among the analysis classes. The refinement of these initial classes into programming level objects comes later. Nonetheless, the structural model at this point should represent the responsibilities of each class and the collaborations between the classes. Typically, structural models are depicted using CRC cards, class diagrams, and, in some cases, object diagrams. However, before describing CRC cards, class diagrams, and object diagrams, we discuss the basic elements of structural models: classes, attributes, operations, and relationships.

Classes, Attributes, and Operations

As stated in Chapter 2, a *class* is a general template that we use to create specific *instances*, or *objects*, in the application domain. All objects of a given class are identical in structure and behavior but contain different data in their attributes. There are two different general kinds of classes of interest during the analysis phase: concrete and abstract. Normally, when analysts describe the application domain classes, they are referring to concrete classes; that is, *concrete classes* are used to create objects. *Abstract classes*, on the other hand, simply are

useful abstractions. For example, from an employee class and a customer class, we may identify a generalization of the two classes and name the abstract class person. We may not actually instantiate the Person class in the system itself, but rather only create and use employees and customers. We will describe generalization in more detail with relationships later in this chapter.

A second classification of classes is the type of real-world thing that it represents. There are application domain classes, user interface classes, data structure classes, file structure classes, operating environment classes, document classes, and various types of multimedia classes. At this point in the development of our evolving system, we are interested only in application domain classes. Later in design and implementation, the other types of classes will become more relevant.

An *attribute* of an analysis class represents a piece of information that is relevant to the description of the class within the application domain of the problem being investigated. An attribute contains information that the analyst or user feels the system should store. For example, a possible relevant attribute of an employee class is employee name, while one that may not be as relevant is hair color. Both describe something about an employee, but hair color probably is not all that useful for most business applications. Only those attributes that are important to the task should be included in the class. Finally, one should only add attributes that are primitive or atomic types, that is, integers, strings, doubles, date, time, boolean, and so on. Most complex or compound attributes are really placeholders for relationships between classes. Therefore, they should be modeled as relationships, not attributes.

An *operation*, or service, of an analysis class is where the *behavior* of the class will be defined. In later phases, the operations will be converted to methods (see Chapter 2). However, since methods are more related to implementation, at this point in the development we use the term operation to describe the actions to which the instances of the class will be capable of responding. Like attributes, only problem domain-specific operations that are relevant to the problem being investigated should be considered. For example, classes are normally required to provide the means to create instances, delete instances, access individual attribute values, set individual attribute values, access individual relationship values, and remove individual relationship values. However, at this point in the development of the evolving system, the analyst should avoid cluttering up the definition of the class with these basic types of operations and only focus on relevant problem domain-specific operations.

Relationships

Many different types of relationships can be defined, but all can be classified into three basic categories of data abstraction mechanisms: *generalization relationships*, *aggregation relationships*, and *association relationships*. These data abstraction mechanisms allow the analyst to focus on the important dimensions while ignoring nonessential dimensions. Like attributes, the analyst must be careful to include only relevant relationships.

Generalization Relationships The generalization abstraction enables the analyst to create classes that inherit attributes and operations of other classes. The analyst creates a *superclass* that contains the basic attributes and operations that will be used in several *subclasses*. The subclasses inherit the attributes and operations of their superclass and can also contain attributes and operations that are unique just to them. For example, a customer class and an employee class can be generalized into a person class by extracting the attributes and operations they have in common and placing them into the new superclass, Person. In this way, the analyst can reduce the redundancy in the class definitions so that the common elements are defined once and then reused in the subclasses. Generalization is represented with the *a-kind-of* relationship, so that we say that an employee is a-kind-of person.

The analyst also can use the flip side of generalization, *specialization*, to uncover additional classes by allowing new subclasses to be created from an existing class. For example, an employee class can be specialized into a secretary class and an engineer class. Furthermore, generalization relationships between classes can be combined to form generalization hierarchies. Based on the previous examples, a secretary class and an engineer class can be subclasses of an employee class, which in turn could be a subclass of a person class. This would be read as, “a secretary and an engineer are a-kind-of employee and a customer and an employee are a-kind-of person.”

The generalization data abstraction is a very powerful mechanism that encourages the analyst to focus on the properties that make each class unique by allowing the similarities to be factored into superclasses. However, to ensure that the semantics of the subclasses are maintained, the analyst should apply the principle of *substitutability*. By this we mean that the subclass should be capable of substituting for the superclass anywhere that uses the superclass (e.g., anywhere we use the employee superclass, we could also logically use its secretary subclass). By focusing on the a-kind-of interpretation of the generalization relationship, the principle of substitutability will be applied.

Aggregation Relationships Many different types of aggregation or composition relationships have been proposed in data modeling, knowledge representation, and linguistics. For example, a-part-of (logically or physically), a-member-of (as in set membership), contained-in, related-to, and associated-with. However, generally speaking, all aggregation relationships relate parts to *wholes* or *parts* to *assemblies*. For our purposes, we use the *a-part-of* or *has-parts* semantic relationship to represent the aggregation abstraction. For example, a door is a-part-of a car, an employee is a-part-of a department, or a department is a-part-of an organization. Like the generalization relationship, aggregation relationships can be combined into aggregation hierarchies. For example, a piston is a-part-of an engine, while an engine is a-part-of a car.

Aggregation relationships are bidirectional in nature. The flip side of aggregation is *decomposition*. The analyst can use decomposition to uncover parts of a class that should be modeled separately. For example, if a door and engine are a-part-of a car, then a car has-parts door and engine. The analyst can bounce around between the various parts to uncover new parts. For example, the analyst can ask, “What other parts are there to a car?” or “To which other assemblies can a door belong?”

Association Relationships There are other types of relationships that do not fit neatly into a generalization (a-kind-of) or aggregation (a-part-of) framework. Technically speaking, these relationships are usually a weaker form of the aggregation relationship. For example, a patient schedules an appointment. It could be argued that a patient is a-part-of an appointment. However, there is a clear semantic difference between this type of relationship and one that models the relationship between doors and cars or even workers and unions. As such, they are simply considered to be associations between instances of classes.

CLASS-RESPONSIBILITY-COLLABORATION CARDS

Class-Responsibility-Collaboration cards, most commonly called *CRC cards*, are used to document the responsibilities and collaborations of a class. We use an extended form of the CRC card to capture all relevant information associated with a class.¹ We describe the elements of our CRC cards below after we explain responsibilities and collaborations.

¹ Our CRC cards are based on the work of D. Bellin and S.S. Simone, *The CRC Card Book* (Reading, MA: Addison-Wesley, 1997), I. Graham, *Migrating to Object Technology* (Wokingham, England: Addison-Wesley, 1995), and B. Henderson-Sellers and B. Unhelkar *OPEN Modeling with UML* (Harlow, England: Addison-Wesley, 2000).

Responsibilities and Collaborations

The *responsibilities* of a class can be divided into two separate types: knowing and doing. *Knowing responsibilities* are those things that an instance of a class must be capable of knowing. An instance of a class typically knows the values of its attributes and its relationships. *Doing responsibilities* are those things that an instance of a class must be capable of doing. In this case, an instance of a class can execute its operations, or it can request a second instance that it knows about to execute one of its operations on behalf of the first instance.

The structural model describes the objects necessary to support the business processes modeled by the use cases. Most use cases involve a set of several classes, not just one class. This set of classes forms a *collaboration*. Collaborations allow the analyst to think in terms of clients, servers, and contracts.² A *client* object is an instance of a class that sends a request to an instance of another class for an operation to be executed. A *server* object is the instance that receives the request from the client object. A *contract* formalizes the interactions between the client and server objects. For example, a patient makes an appointment with a doctor. This sets up an obligation for both the patient and doctor to appear at the appointed time. Otherwise, consequences, such as billing the patient for the appointment regardless of whether the patient appears, can be dealt out. The contract should also spell out what the benefits of the contract will be, such as a treatment being prescribed for whatever ails the patient and a payment to the doctor for the services provided. Chapter 10 provides a more detailed explanation of contracts and examples of their use.

The analyst can use the idea of class responsibilities and client-server-contract collaborations to help identify the classes, along with the attributes, operations, and relationships, involved with a use case. One of the easiest ways in which to use CRC cards in developing a structural model is through *anthropomorphism*—pretending that the classes have human characteristics. This is done by having the analyst and/or the user *role-play* and pretend that they are an instance of the class being considered. They then can either ask questions of themselves or be asked questions by other members of the development team. For example,

Who or what are you?
What do you know?
What can you do?

The answers to the questions are then used to add detail to the evolving CRC cards.

Elements of a CRC Card

The set of CRC cards contains all the information necessary to build a logical structural model of the problem under investigation. Figure 7-1 shows a sample CRC card. Each card captures and describes the essential elements of a class. The front of the card allows the recording of the class's name, ID, type, description, list of associated use cases, responsibilities, and collaborators. The name of a class should be a noun (but not a proper noun such as the name of a specific person or thing). Just like the use cases, in later stages of development, it is important to be able to trace back design decisions to specific requirements. In

² For more information, see K. Beck and W. Cunningham, "A Laboratory for Teaching Object-Oriented Thinking," *Proceedings of OOPSLA, SIGPLAN Notices* 24, no. 10 (1989): 1–6; Henderson-Sellers and Unhelkar, *OPEN Modeling with UML*; C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, (Englewood Cliffs, NJ: Prentice-Hall, 1998); B. Meyer, *Object-Oriented Software Construction* (Englewood Cliffs, NJ: Prentice Hall, 1994); and R. Wirfs-Brock, B. Wilkerson, and L. Wiener, *Designing Object-Oriented Software* (Englewood Cliffs, NJ: Prentice Hall, 1990).

TEMPLATE
can be found at
[www.wiley.com/
college/dennis](http://www.wiley.com/college/dennis)

Front:		
Class Name: Patient	ID: 3	Type: Concrete, Domain
Description: An Individual that needs to receive or has received medical attention		Associated Use Cases: 2
Responsibilities Make appointment Calculate last visit Change status Provide medical history _____ _____ _____ _____		Collaborators Appointment _____ _____ Medical history _____ _____ _____ _____
Back:		
Attributes: Amount (double) Insurance carrier (text) _____ _____		
Relationships: Generalization (a-kind-of): Person _____		
Aggregation (has-parts): Medical History _____		
Other Associations: Appointment _____		

FIGURE 7-1
Example CRC Card

conjunction with the list of associated use cases, the ID number for each class can be used to accomplish this. The description is simply a brief statement that can be used as a textual definition for the class. The responsibilities of the class tend to be the operations that the class must contain, that is, the doing responsibilities.

The back of a CRC card contains the attributes and relationships of the class. The attributes of the class represent the knowing responsibilities that each instance of the class will have to meet. Typically, the data type of each attribute is listed with the name of the attribute; for example, the amount attribute is double, and the insurance carrier is text. Three types of relationships are typically captured at this point in time: generalization, aggregation, and other associations. In Figure 7-1, we see that a Patient is a-kind-of Person and that a Patient is associated with Appointments.

Again, CRC cards are used to document the essential properties of a class. However, once the cards are filled out, the analyst can use the cards and anthropomorphism in role-playing to uncover missing properties.

CLASS DIAGRAMS

The *class diagram* is a *static model* that shows the classes and the relationships among classes that remain constant in the system over time. The class diagram depicts classes, which include both behaviors and states, with the relationships between the classes. The following sections will first present the elements of the class diagram, followed by the way in which a class diagram is drawn.

Elements of a Class Diagram

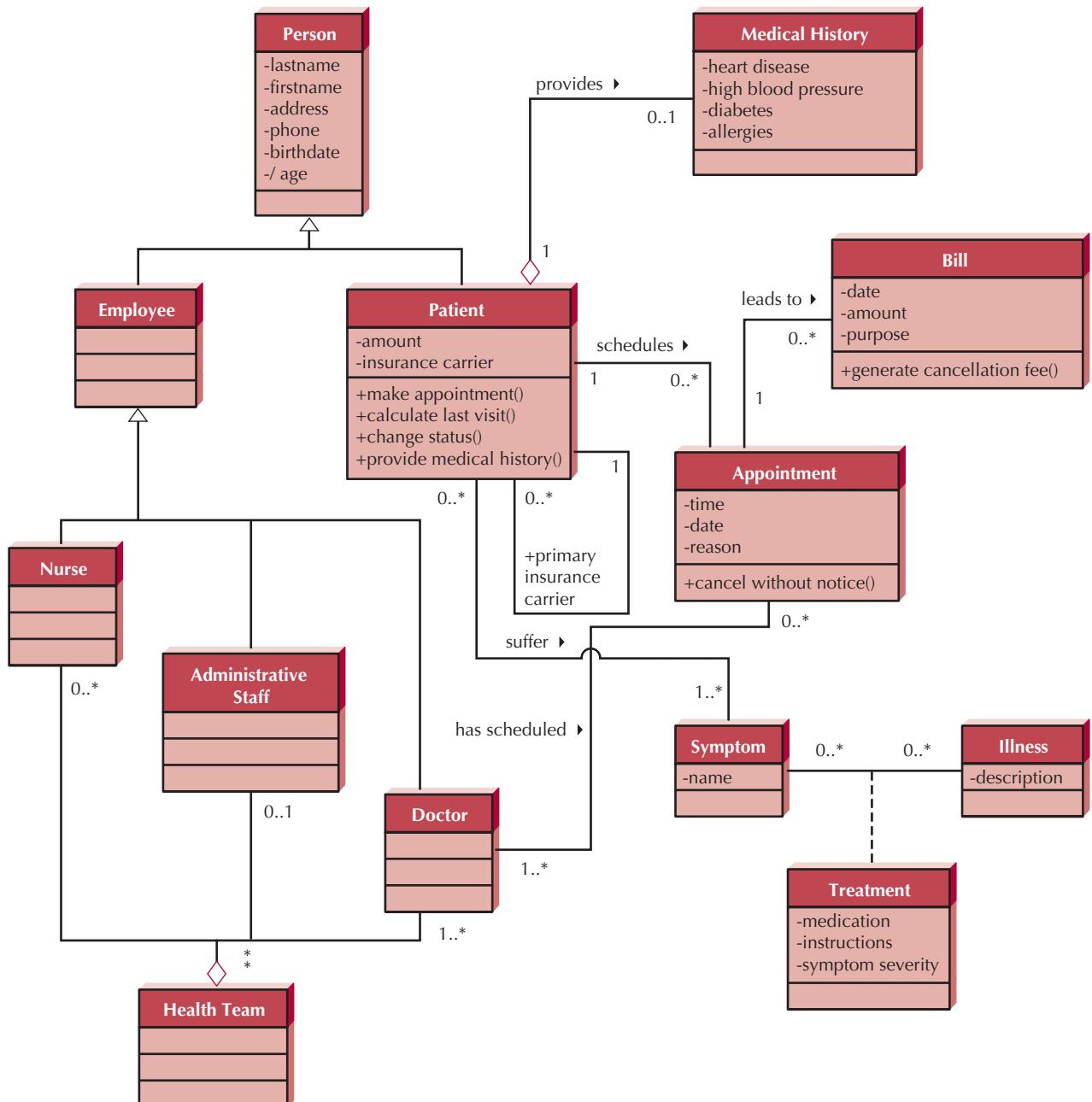
Figure 7-2 shows a class diagram that was created to reflect the classes and relationships that are needed for the set of use cases that describe the appointment system in Chapter 6.

Class The main building block of a *class diagram* is the class, which stores and manages information in the system (see Figure 7-3). During analysis, classes refer to the people, places, events, and things about which the system will capture information. Later, during design and implementation, classes can refer to implementation-specific artifacts like windows, forms, and other objects used to build the system. Each class is drawn using three-part rectangles with the class's name at the top, attributes in the middle, and operations at the bottom. For example, you should be able to identify Person, Doctor, Patient, Medical History, Appointment, Bill, and Symptom as some of the classes in Figure 7-2. The attributes of a class and their values define the state of each object that is created from the class, and the behavior is represented by the operations.

Attributes are properties of the class about which we want to capture information (see Figure 7-3). Notice that the Person class in Figure 7-2 contains the attributes lastname, firstname, address, phone, and birthdate. At times, you may want to store *derived attributes*, which are attributes that can be calculated or derived; these special attributes are denoted by placing a slash (/) before the attribute's name. Notice how the Person class contains a derived attribute called "/age," which can be derived by subtracting the patient's birthdate from the current date. It is also possible to show the *visibility* of the attribute on the diagram. *Visibility* relates to the level of information hiding (see Chapter 2) to be enforced for the attribute. The visibility of an attribute can be public (+), protected (#), or private (-). A *public* attribute is one that is not hidden from any other object. As such, other objects can modify its value. A *protected* attribute is one that is hidden from all other classes except its immediate subclasses. A *private* attribute is one that is hidden from all other classes. The default visibility for an attribute is normally private.

Operations are actions or functions that a class can perform (see Figure 7-3). The functions that are available to all classes (e.g., create a new instance, return a value for a particular attribute, set a value for a particular attribute, or delete an instance) are not explicitly shown within the class rectangle. Instead, only those operations that are unique to the class are included, such as the "cancel without notice" operation in the Appointment class and the "generate cancellation fee" operation in the Bill class in Figure 7-2. Notice that both of the operations are followed by parentheses that contain the parameter(s) needed by the operation. If an operation has no parameters, the parentheses are still shown but are empty. As with attributes, the visibility of an operation can be designated as public, protected, or private. The default visibility for an operation is normally public.

A class can contain three kinds of operations: constructor, query, and update. A *constructor operation* creates a new instance of a class. For example, the patient class may have a method called "insert ()" that creates a new patient instance as patients are entered into the system. As we just mentioned, if a method is available to all classes (e.g., create a new instance), it is not explicitly shown on the class diagram, so typically you will not see constructor methods explicitly on the class diagram.

**FIGURE 7-2** Example Class Diagram

A *query operation* makes information about the state of an object available to other objects, but it will not alter the object in any way. For instance, the “calculate last visit ()” operation that determines when a patient last visited the doctor’s office will result in the object being accessed by the system, but it will not make any change to its information. If a query method merely asks for information from attributes in the class (e.g., a patient’s name, address, or phone), then it is not shown on the diagram because we assume that all objects have operations that produce the values of their attributes.

A CLASS	<ul style="list-style-type: none"> Represents a kind of person, place, or thing about which the system will need to capture and store information Has a name typed in bold and centered in its top compartment Has a list of attributes in its middle compartment Has a list of operations in its bottom compartment Does not explicitly show operations that are available to all classes 	<pre> classDiagram class Class1 { -attribute1 +operation1() } </pre>
AN ATTRIBUTE	<ul style="list-style-type: none"> Represents properties that describe the state of an object Can be derived from other attributes, shown by placing a slash before the attribute's name 	attribute name /derived attribute name
AN OPERATION	<ul style="list-style-type: none"> Represents the actions or functions that a class can perform Can be classified as a constructor, query, or update operation Includes parentheses that may contain parameters or information needed to perform the operation 	operation name ()
AN ASSOCIATION	<ul style="list-style-type: none"> Represents a relationship between multiple classes, or a class and itself Is labeled using a verb phrase or a role name, whichever better represents the relationship Can exist between one or more classes Contains multiplicity symbols, which represent the minimum and maximum times a class instance can be associated with the related class instance 	<pre> classDiagram "1..*" --> "0..1" : verb phrase </pre>

FIGURE 7-3 Class Diagram Syntax

An *update operation* will change the value of some or all of the object's attributes, which may result in a change in the object's state. Consider changing the status of a patient from new to current with a method called “change status ()”, or associating a patient with a particular appointment with “schedule appointment (appointment).”

Relationships A primary purpose of the class diagram is to show the relationships, or *associations*, that classes have with one another. These are depicted on the diagram by drawing lines between classes (see Figure 7-3). When multiple classes share a relationship (or a class shares a relationship with itself), a line is drawn and labeled with either the name of the relationship or the roles that the classes play in the relationship. For example, in Figure 7-2 the two classes Patient and Appointment are associated with one another whenever a patient schedules an appointment. Thus, a line labeled “schedules” connects patient and appointment, representing exactly how the two classes are related to each other. Also notice that there is a small solid triangle beside the name of the relationship. The triangle allows a direction to be associated with the name of the relationship. In Figure 7-2, the schedules relationship includes a triangle, indicating that the relationship is to be read as “patient schedules appointment.” Inclusion of the triangle simply increases the readability of the diagram.

Sometimes a class is related to itself, as in the case of a patient being the primary insurance carrier for other patients (e.g., their spouse, children). In Figure 7-2, notice that a line was drawn between the Patient class and itself, and is called “primary insurance carrier” to depict the role that the class plays in the relationship. Notice that a plus “+” sign is placed before the label to communicate that it is a role as opposed to the name of the relationship. When labeling an association, use either a relationship name or a role name (not both), whichever communicates a more thorough understanding of the model.

Relationships also have *multiplicity*, which documents how an instance of an object can be associated with other instances. Numbers are placed on the association path to denote the minimum and maximum instances that can be related through the association in the format *minimum number . . . maximum number* (see Figure 7-4). The numbers specify the relationship from the class at the far end of the relationship line to the end with the number. For example, in Figure 7-2, there is a “0..*” on the appointment end of the “patient schedules appointment” relationship. This means that a patient can be associated with zero through many different appointments. At the patient end of this same relationship there is a “1,” meaning that an appointment must be associated with one and only one (1) patient.

There are times when a relationship itself has associated properties, especially when its classes share a many-to-many relationship. In these cases, a class is formed, called an *association class*, that has its own attributes and operations. It is shown as a rectangle attached by a dashed line to the association path, and the rectangle’s name matches the label of the association. Think about the case of capturing information about illnesses and symptoms. An illness (e.g., the flu) can be associated with many symptoms (e.g., sore throat, fever), and a symptom (e.g., sore throat) can be associated with many illnesses (e.g., the flu, strep throat, the common cold). Figure 7-2 shows how an association class can capture information about remedies that change depending on the various combinations. For example, a sore throat caused by the strep bacterium will require antibiotics, whereas treatment for a sore throat from the flu or a cold could be throat lozenges or hot tea. Most often, classes are related through a “normal” association; however, two special cases of an association will appear quite often: generalization and aggregation.

Generalization and Aggregation Generalization shows that one class (subclass) inherits from another class (superclass), meaning that the properties and operations of the superclass are also valid for objects of the subclass. The generalization path is shown with a solid line from the subclass to the superclass and a hollow arrow pointing at the superclass. For example, Figure 7-2 tells us that doctors, nurses, and administrative personnel are all kinds of employees and that those employees and patients are kinds of persons. Remember that the generalization relationship occurs when you need to use words such as “is a-kind-of” to describe the relationship.

Exactly one	1	<pre> classDiagram class Department class Boss Department "1" --> "1" Boss </pre>	A department has one and only one boss.
Zero or more	0..*	<pre> classDiagram class Employee class Child Employee "0..*" --> "0..*" Child </pre>	An employee has zero to many children.
One or more	1..*	<pre> classDiagram class Boss class Employee Boss "1..*" --> "1..*" Employee </pre>	A boss is responsible for one or more employees.
Zero or one	0..1	<pre> classDiagram class Employee class Spouse Employee "0..1" --> "0..1" Spouse </pre>	An employee can be married to zero or no spouse.
Specified range	2..4	<pre> classDiagram class Employee class Vacation Employee "2..4" --> "2..4" Vacation </pre>	An employee can take between two to four vacations each year.
Multiple, disjoint ranges	1..3, 5	<pre> classDiagram class Employee class Committee Employee "1..3, 5" --> "1..3, 5" Committee </pre>	An employee is a member of one to three or five committees.

FIGURE 7-4 Multiplicity

Aggregation is used when classes actually comprise other classes. For example, think about a doctor's office that has decided to create health care teams that include doctors, nurses, and administrative personnel. As patients enter the office, they are assigned to a health care team that cares for their needs during their visits. Figure 7-2 shows how this relationship is denoted on the class diagram. A diamond is placed nearest the class representing the aggregation (health care team), and lines are drawn from the arrow to connect the classes that serve as its parts (doctors, nurses, and administrative staff). Typically, you can identify these kinds of associations when you need to use words like "is a part of" or "is made up of" to describe the relationship.

Simplifying Class Diagrams

When a class diagram is fully populated with all of the classes and relationships for a real-world system, the class diagram can become very difficult to interpret, that is, very complex. When this occurs, it is sometimes necessary to simplify the diagram. Using a *view* mechanism can simplify the class diagram. Views were developed originally with relational database management systems and are simply a subset of the information contained in the database. In this case, the view would be a useful subset of the class diagram, such as a use-case view that shows only the classes and relationships that are relevant to a particular use case. A second view could be to show only a particular type of relationship: aggregation, association, or generalization. A third type of view is to restrict the information shown with each class, for example, only show the name of the class, the name and attributes, or the name and operations. These view mechanisms can be combined to further simplify the diagram.

A second approach to simplify a class diagram is through the use of *packages* (i.e., logical groups of classes). To make the diagrams easier to read and to keep the models at a reasonable level of complexity, you can group the classes together into packages. Packages are general constructs that can be applied to any of the elements in UML models. In Chapter 6, we introduced them to simplify use-case diagrams. In the case of class diagrams, it is simple to sort the classes into groups based on the relationships they share.³

Object Diagrams

Although class diagrams are necessary to document the structure of the classes, at times, an *object diagram*, can be useful. An object diagram is essentially an instantiation of all or part of a class diagram. (*Instantiation* means to create an instance of the class with a set of appropriate attribute values.)

Object diagrams can be very useful when one is trying to uncover details of a class. Generally speaking, it is easier to think in terms of concrete objects (instances) rather than abstractions of objects (classes). For example, in Figure 7-5, a portion of the class diagram in Figure 7-2 has been copied and instantiated. The top part of the figure simply is a copy of a small view of the overall class diagram. The lower portion is the object diagram that instantiates that subset of classes. By reviewing the actual instances involved, John Doe, Appt1, and Dr. Smith, we may discover additional relevant attributes, relationships, and/or operations or possibly misplaced attributes, relationships, and/or operations. For example, an appointment has a reason attribute. Upon closer examination, the reason attribute may have been better modeled as an association with the Symptom class. Currently, the Symptom class is associated with the Patient class. After reviewing the object diagram, this seems to be in error. As such, we should modify the class diagram to reflect this new understanding of the problem.

³ For those familiar with structured analysis and design, packages serve a similar purpose as the leveling and balancing processes used in data flow diagramming. Packages and package diagrams are described in more detail in Chapter 9.

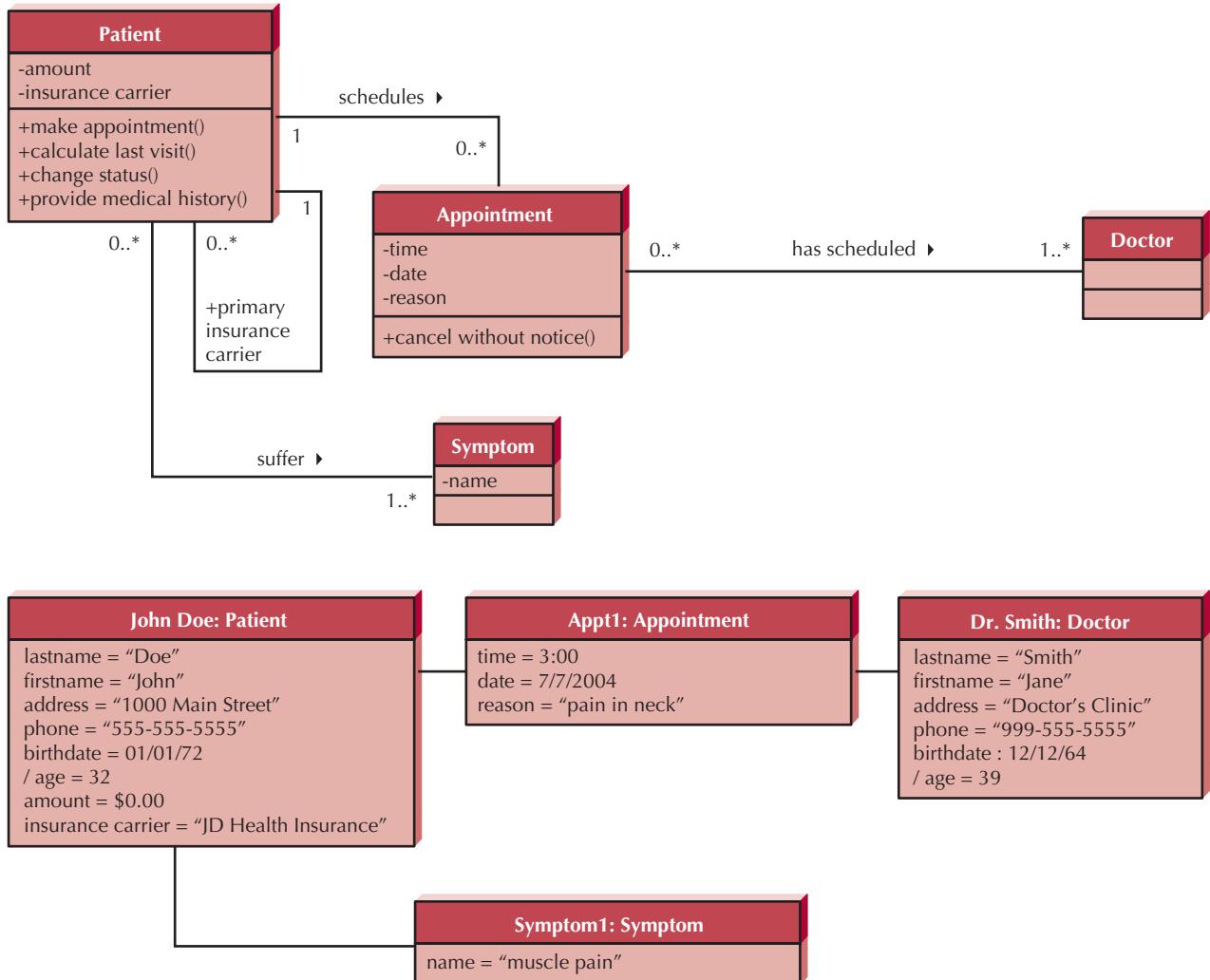


FIGURE 7-5 Example Object Diagram

CREATING CRC CARDS AND CLASS DIAGRAMS

Creating a structural model is an iterative process whereby the analyst makes a roughcut of the model and then refines it over time. Structural models can become quite complex. In fact, some systems have class diagrams that contain hundreds of classes. In this section, we take you through one iteration of creating a structural model, but the model can be expected to change dramatically as you communicate with users and fine-tune your understanding of the system. First, however, we present a set of approaches to identify and refine a set of candidate objects, and provide a set of steps based on these approaches that can be used to create the initial roughcut structural model.

Object Identification

Many different approaches have been suggested to aid the analyst in identifying a set of candidate objects for the structural model. The three most common approaches are textual analysis, common object lists, and patterns. Most analysts use a combination of all three

techniques to make sure that no important objects and object attributes, operations, and relationships have been overlooked.

Textual Analysis *Textual analysis* is an analysis of the text in the use-case descriptions. The analyst starts by reviewing the use-case descriptions and the use-case diagrams. The text in the descriptions is examined to identify potential objects, attributes, operations, and relationships. The nouns in the use case suggest possible classes, whereas the verbs suggest possible operations. Figure 7-6 presents a summary of guidelines we have found useful. The textual analysis of use-case descriptions has been criticized as being too simple, but because its primary purpose is to create an initial roughcut structural model, its simplicity is a major advantage.

Common Object List As its name implies, a *common object list* is simply a list of objects that are common to the business domain of the system. In addition to looking at the specific use cases, the analyst also thinks about the business separately from the use cases. Several categories of objects have been found to help the analyst in creation of the list, such as physical or tangible things, incidents, roles, and interactions.⁴ Analysts should first look for physical or *tangible things* in the business domain. These could include books, desks, chairs, and office equipment. Normally, these types of objects are the easiest to identify. *Incidents* are events that occur in the business domain, such as meetings, flights, performances, or accidents. Reviewing the use cases can readily identify the *roles* that the people “play” in the problem, such as doctor, nurse, patient, or receptionist. Typically, an *interaction* is a transaction that takes place in the business domain, such as a sales transaction. Other types of objects that can be identified include places, containers, organizations, business records, catalogs, and policies. In rare cases, processes themselves may need information stored about them. In these cases, processes may need an object, in addition to a use case, to represent them.

- A common or improper noun implies a class of objects.
- A proper noun or direct reference implies an instance of a class.
- A collective noun implies a class of objects made up of groups of instances of another class.
- An adjective implies an attribute of an object.
- A doing verb implies an operation.
- A being verb implies a classification relationship between an object and its class.
- A having verb implies an aggregation or association relationship.
- A transitive verb implies an operation.
- An intransitive verb implies an exception.
- A predicate or descriptive verb phrase implies an operation.
- An adverb implies an attribute of a relationship or an operation.

Source: These guidelines are based on Russell J. Abbott, “Program Design by Informal English Descriptions,” *Communications of the ACM* 26, no. 11 (1983): 882–894; Peter P-S Chen, “English Sentence Structure and Entity-Relationship Diagrams,” *Information Sciences: An International Journal* 29, no. 2–3 (1983): 127–149; and Graham, *Migrating to Object Technology*.

FIGURE 7-6
Textual Analysis
Guidelines

⁴ For example, see Larman, *Applying UML and Patterns*, and Sally Shlaer and Stephen J. Mellor, *Object-Oriented Systems Analysis: Modeling the World in Data* (Englewood Cliffs, NJ: Yourdon Press, 1988).

Patterns The idea of using patterns is relatively new in object-oriented systems development.⁵ There have been many definitions of exactly what a pattern is. From our perspective, a *pattern* is simply a useful group of collaborating classes that provide a solution to a commonly occurring problem. Since patterns provide a solution to commonly occurring problems, they are reusable.

An architect, Christopher Alexander, has inspired much of the work associated with using patterns in object-oriented systems development. According to Alexander and his colleagues,⁶ very sophisticated buildings can be constructed by stringing together commonly found patterns, rather than creating entirely new concepts and designs. In a very similar manner, it is possible to put together commonly found object-oriented patterns to form elegant object-oriented information systems. For example, many business transactions involve the same type of objects and interactions. Virtually all transactions would require a transaction class, a transaction line item class, an item class, a location class, and a participant class. By simply reusing these existing patterns of classes, we can define the system more quickly and more completely than by starting with a blank piece of paper.

Many different types of patterns have been proposed, ranging from high-level business-oriented patterns to more low-level design patterns. Figure 7-7 lists some common business domains for which patterns have been developed, as well as their source. If you are developing a business information system in one of these business domains, then the patterns developed for that domain may be a very useful starting point in identifying needed classes and their attributes, operations, and relationships.

Building CRC Cards and Class Diagrams

It is important to remember that although CRC cards and class diagrams can be used to describe both the As-Is and To-Be structural model of the evolving system, they are most often used for the To-Be model. There are many different ways to identify a set of candidate objects and to create CRC cards and class diagrams. Today most object identification begins with the use cases identified for the problem (see Chapter 6). In this section, we describe a seven-step process used to create the structural model of the problem (see Figure 7-8).

Create CRC Cards We could begin creating the structural model with a class diagram instead of CRC cards. However, due to the ease of role playing with CRC cards, we prefer to create the CRC cards first and then transfer the information from the CRC cards into a class diagram later. As a result, the first step of our recommended process is to create CRC cards. This is done by performing textual analysis on the use case descriptions. If you recall, the normal flow of events, subflows, and alternate/exceptional flows written as part of the use case were written in a special form called Subject Verb Direct object Preposition Indirect object (*SVDPI*). By writing use-case events in this form, it is

⁵ Many books have been devoted to this topic: for example, see Peter Coad, David North, and Mark Mayfield, *Object Models: Strategies, Patterns, & Applications*, 2nd ed., (Englewood Cliffs, NJ: Prentice Hall, 1997); Hans-Erik Eriksson and Magnus Penker, *Business Modeling with UML: Business Patterns at Work*, (New York: John Wiley & Sons, 2000); Martin Fowler, *Analysis Patterns: Reusable Object Models*, (Reading, MA: Addison-Wesley, 1997); Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, (Reading, MA: Addison-Wesley, 1995); and David C. Hay, *Data Model Patterns: Conventions of Thought* (New York, NY: Dorset House, 1996).

⁶ Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel, *A Pattern Language* (New York: Oxford University Press, 1977).

easier to use the guidelines for use-case analysis in Figure 7-6 to identify the objects. Reviewing the primary actors, stakeholders, and interests, and giving a brief description of each use case allow additional candidate objects to be identified. Furthermore, it is useful to go back and review the original requirements to look for information that was not included in the text of the use cases. Record all information uncovered for each candidate object on a CRC card.

Business Domains	Sources of Patterns
Accounting	3, 4
Actor-Role	2
Assembly-Part	1
Container-Content	1
Contract	2, 4
Document	2, 4
Employment	2, 4
Financial Derivative Contracts	3
Geographic Location	2, 4
Group-Member	1
Interaction	1
Material Requirements Planning	4
Organization and Party	2, 3
Plan	1, 3
Process Manufacturing	4
Trading	3
Transactions	1, 4
1. Peter Coad, David North, and Mark Mayfield, <i>Object Models: Strategies, Patterns, & Applications</i> , 2nd ed., (Englewood Cliffs, NJ: Prentice Hall, 1997). 2. Hans-Erik Eriksson and Magnus Penker, <i>Business Modeling with UML: Business Patterns at Work</i> , (New York: John Wiley & Sons, 2000). 3. Martin Fowler, <i>Analysis Patterns: Reusable Object Models</i> (Reading, MA: Addison-Wesley, 1997). 4. David C. Hay, <i>Data Model Patterns: Conventions of Thought</i> (New York, NY, Dorset House, 1996).	

FIGURE 7-7
Useful Patterns

1. Create CRC cards by performing textual analysis on the use cases.
2. Brainstorm additional candidate classes, attributes, operations, and relationships by using the common object list approach.
3. Role-play each use case using the CRC cards.
4. Create the class diagram based on CRC cards.
5. Review the structural model for missing and/or unnecessary classes, attributes, operations, and relationships.
6. Incorporate useful patterns.
7. Review the structural model.

FIGURE 7-8
Steps for Object
Identification and
Structural Modeling

Examine Common Object Lists The second step is to brainstorm additional candidate objects, attributes, operations, and relationships using the common object list approach. What are the tangible things associated with the problem? What are the roles played by the people in the problem domain? What incidents and interactions take place in the problem domain? As you can readily see, by beginning with the use cases, many of these questions already have partial answers to them. For example, the primary actors and stakeholders are the roles that are played by the people in the problem domain. However, it is possible to uncover additional roles that were not thought of previously. This obviously would cause the use cases and the use-case model to be modified and possibly expanded. As in the previous step, be sure to record all information uncovered onto the CRC cards. This includes any modifications uncovered for any previously identified candidate objects and any information regarding any new candidate objects identified.

Role-Play the CRC Cards The third step is to role-play each use case using the CRC cards.⁷ Each CRC Card should be assigned to an individual, who will perform the operations for the class on the CRC card. As the “performers” play out their roles, the “system” will tend to break down. When this occurs, additional objects, attributes, operations, or relationships will be identified. Again, like the previous steps, anytime any new information is discovered, new CRC cards are created or modifications to existing CRC cards are made.

Create the Class Diagram The fourth step is to create the class diagram based on the CRC cards. This is equivalent to creating the use-case diagram from the use case descriptions. Information contained on the CRC cards is simply transferred to the class diagrams. The responsibilities are transferred as operations, and the attributes as attributes, and the relationships are drawn as generalization, aggregation, or association relationships. However, the class diagram also requires that the visibility of the attributes and operations be known. As a general rule of thumb, attributes are private and operations are public. Therefore, unless the analyst has a good reason to change the default visibility of these properties, then the defaults should be accepted. Finally, the analyst should examine the model for additional opportunities to use aggregation or generalization relationships. These types of relationships can simplify the individual class descriptions. As in the previous steps, any and all changes must be recorded on the CRC cards.

Review the Class Diagram The fifth step is to review the structural model for missing and/or unnecessary classes, attributes, operations, and relationships. Up until this step, the focus of the process has been on adding information to the evolving model. At this point in time, the focus begins to switch from only adding information to also challenging the reasons for including the information contained in the model.

Incorporate Patterns The sixth step is to incorporate useful patterns into the evolving structural model. A useful pattern is one that would allow the analyst to more fully describe the underlying domain of the problem being investigated. We can do so by looking at the collection of patterns available (Figure 7-7) and comparing the classes contained in the patterns with those in the evolving class diagram. After identifying the useful patterns, the analyst incorporates the identified patterns into the class diagram and modifies the affected CRC cards. This includes adding and removing classes, attributes, operations, and/or relationships.

⁷ For more information on role-playing, see Bellin and Simone, *The CRC Card Book* and Dean Leffingwell and Don Widrig, *Managing Software Requirements: A Unified Approach* (Reading, MA: Addison-Wesley, 2000).

Review the Model The seventh and final step is to review the structural model, including both the CRC cards and the class diagram. This is best accomplished during a formal review meeting using a walkthrough approach in which an analyst presents the model to a team of developers and users. The analyst “walks through” the model explaining each part of the model and all of the reasoning that went into the decision to include each class in the structural model. This explanation includes justifications for the attributes, operations, and relationships associated with the classes. Finally, each class should be linked back to at least one use case; otherwise, the purpose of including the class in the structural model will not be understood. It is often best for the review team to also include people outside the development team that produced the model because these individuals can bring a fresh perspective to the model and uncover missing objects.

Applying the Concepts at CD Selections

The previous chapter described the CD Selections Internet sales system (see Figures 6-12 through 6-18). In this section, we demonstrate the process of structural modeling using one of the use cases identified: Place Order (see Figure 6-15). Even though we are using just one of the use cases for our structural model, you should remember that to create a complete structural model all use cases should be used.

Create CRC Cards The first step was to create the set of CRC cards by performing textual analysis on the use cases. To begin with, Alec chose the Place Order use case (see Figure 6-15). He and his team then used the textual analysis rules (see Figure 7-6) to identify the candidate classes, attributes, operations, and relationships. Using these rules on the Normal Flow of Events, they identified Customer, Search Request, CD, List, and Review as candidate classes. They uncovered three different types of search requests: Title Search, Artist Search, and Category Search. By applying the textual analysis rules to the Brief Description, an additional candidate class was discovered: Order. By reviewing the verbs contained in this use case, they saw that a Customer places an Order and that a Customer makes a Search Request.

To be as thorough as possible, Alec and his team also reviewed the original requirements used to create the use case. The original requirements are contained in Figure 5-13. After reviewing this information, they identified a set of attributes for the Customer (name, address, e-mail, and credit card) and Order (CDs to purchase and quantity) classes and uncovered additional candidate classes: CD Categories and Credit Card Center. Furthermore, they realized that the Category Search class used the CD Categories class. Finally, they also identified three subclasses of CD Categories: Rock, Jazz, and Classical. Alec’s goal, at this point in time, was to be as complete as possible. As such, he realized that they may have identified many candidate classes, attributes, operations, and relationships that may not be included in the final structural model.

Examine Common Object Lists The second step for Alec and his team was to brainstorm additional candidate classes, attributes, operations, and relationships. He asked the team members to take a minute and think about what information they would like to keep about CDs. The information that they thought of was a set of attributes, for example, title, artist, quantity on hand, price, and category.

He then asked them to take another minute and think about the information that they should store about orders and an order’s responsibilities. The responsibilities they identified were a set of operations, including calculate tax, calculate extension price, calculate shipping, and calculate total. Currently, the attributes (CDs to purchase and quantity) of Order implied that a customer should be allowed to order multiple copies of the same CD

and allow different CDs to be ordered on the same order. However, the current structural model did not allow this. As such, they created a new class that was associated with both the Order class and the CD class: Order Line Item. This new class only had one attribute, quantity, but it had two relationships: one with Order and the other with CD.

When they reviewed the Customer class, they decided that the name and address attributes needed to be expanded; name should become last name, first name, and middle initial, and address should become street address, city, state, country, and zip code. The updated Customer class and Order class CRC cards are shown in Figures 7-9 and 7-10, respectively.

Role-Play the CRC Cards The third step was to role-play the classes recorded on the CRC cards. The purpose of this step was to validate the current state of the evolving structural model. Alec handed out the CRC cards to different members of his team. Using the

Front:												
Class Name: Customer	ID: 1	Type: Concrete, Domain										
Description: An Individual that may or has purchased merchandise from the CD Selections Internet sales system		Associated Use Cases: 3										
Responsibilities <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>		Collaborators <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>										
Back:												
Attributes: <table> <tr> <td>First name</td> <td>State</td> </tr> <tr> <td>Middle Initial</td> <td>Country</td> </tr> <tr> <td>Last name</td> <td>Zip code</td> </tr> <tr> <td>Street address</td> <td>E-mail</td> </tr> <tr> <td>City</td> <td>Credit card</td> </tr> </table>			First name	State	Middle Initial	Country	Last name	Zip code	Street address	E-mail	City	Credit card
First name	State											
Middle Initial	Country											
Last name	Zip code											
Street address	E-mail											
City	Credit card											
Relationships: <p>Generalization (a-kind-of): _____</p> <p>Aggregation (has-parts): _____</p> <p>Other Associations: Order; Search Request _____</p>												

FIGURE 7-9
Customer Class
CRC Card

CRC cards, they began executing the different use cases, one at a time, to see if the current structural model could support each use case or whether the use case caused the “system” to crash. Anytime the “system” crashed, there was something missing: a class, an attribute, a relationship, or an operation. They would then add the missing information to the structural model and try executing the use case again.

First, they decided that the customer had requested the system to perform a search for all of the CDs associated with a specific artist. Based on the current CRC cards, the team

Front:		
Class Name: Order	ID: 2	Type: Concrete, Domain
Description: An Individual that needs to receive or has received medical attention		Associated Use Cases: 3
Responsibilities Calculate tax Calculate shipping Calculate total _____ _____ _____ _____ _____		Collaborators _____ _____ _____ _____ _____ _____ _____
Back:		
Attributes: Tax _____ Shipping _____ Total _____ _____ _____ _____		
Relationships: Generalization (a-kind-of): _____ Aggregation (has-parts): _____ Other Associations: Order Item; Customer _____ _____		

FIGURE 7-10
Order Class
CRC Card

YOUR
TURN

7-1 CD-Selections Internet Sales System

Complete the CRC cards for the remaining identified classes.

felt that the system would produce an accurate list of CDs. They then tried to ask the system for a set of reviews of the CD. At this point in the exercise, the system crashed. The CRC cards did not have the Review class associated with the CD class. Therefore, there was no way to retrieve the requested information. This observation raised another question. Was there other marketing information that should be made available to the customer, for example, artist information and sample clips?

Next, they realized that vendor information should be a separate class that was associated with a CD rather than an additional attribute of a CD. This was because vendors had additional information and operations themselves. If they had modeled the vendor information as an attribute of CD, then the additional information and operations would have been lost. They continued role-playing each of the use cases until they were comfortable with the structural model's ability to support each and every one.

Create the Class Diagram The fourth step was to create the class diagram from the CRC cards. Figure 7-11 shows the current state of the evolving structural model as depicted in a class diagram based on the Place Order use case.

Review the Class Diagram The fifth step was to review the structural model for missing and/or unnecessary classes, attributes, operations, and relationships. At this point, the team challenged all components of the model (class, attribute, relationship, or operation) that did not seem to be adding anything useful to the model. If a component could not be justified sufficiently, then they removed it from the structural model. By carefully reviewing the current state of the structural model, they were able to challenge over a third of the classes contained in the class diagram (see Figure 7-11). It seemed that the CD categories, and their subclasses, were not really necessary. There were no attributes or operations for these classes. As such, the idea of CD categories was modeled as an attribute of a CD. The category attribute for the CD class was previously uncovered during the brainstorming step. Also, upon further review of the Search Request class and its subclasses, it was decided that the subclasses were really nothing more than a set of operations of the Search Request class. This was an example of process decomposition creeping into the modeling process. From an object-oriented perspective, we must always be careful to not allow this to occur. However, during the previous steps in the modeling process, Alec wanted to include as much information as possible in the model. It was more beneficial to remove this type of information after it had crept into the model than to take a chance on not capturing the information required to solve the problem.

Incorporate Patterns The sixth step was to incorporate any useful pattern into the structural model. One pattern that could be useful is the Contract pattern listed in Figure 7-7. By reviewing this pattern (see Figure 7-12), Alec and his team uncovered two subclasses of the Customer class: Individual and Organizational.⁸ Furthermore, Peter Coad has identified twelve transaction patterns that also could be useful for Alec and his team to investigate further.⁹

Review the Model The seventh and final step is to carefully review the structural model. Figure 7-13 shows the Places Order use case view of the structural model as portrayed in a class diagram developed by Alec and his team. This version of the class diagram incorporates all of the modification described previously.

⁸ This pattern is based on the Sales Order contract pattern described in David C. Hay, *Data Model Patterns: Conventions of Thought* (New York, NY: Dorset House, 1996).

⁹ See Peter Coad, David North, and Mark Mayfield, *Object Models: Strategies, Patterns, & Applications*, 2nd Ed. (Englewood Cliffs, NJ: Prentice Hall, 1997).

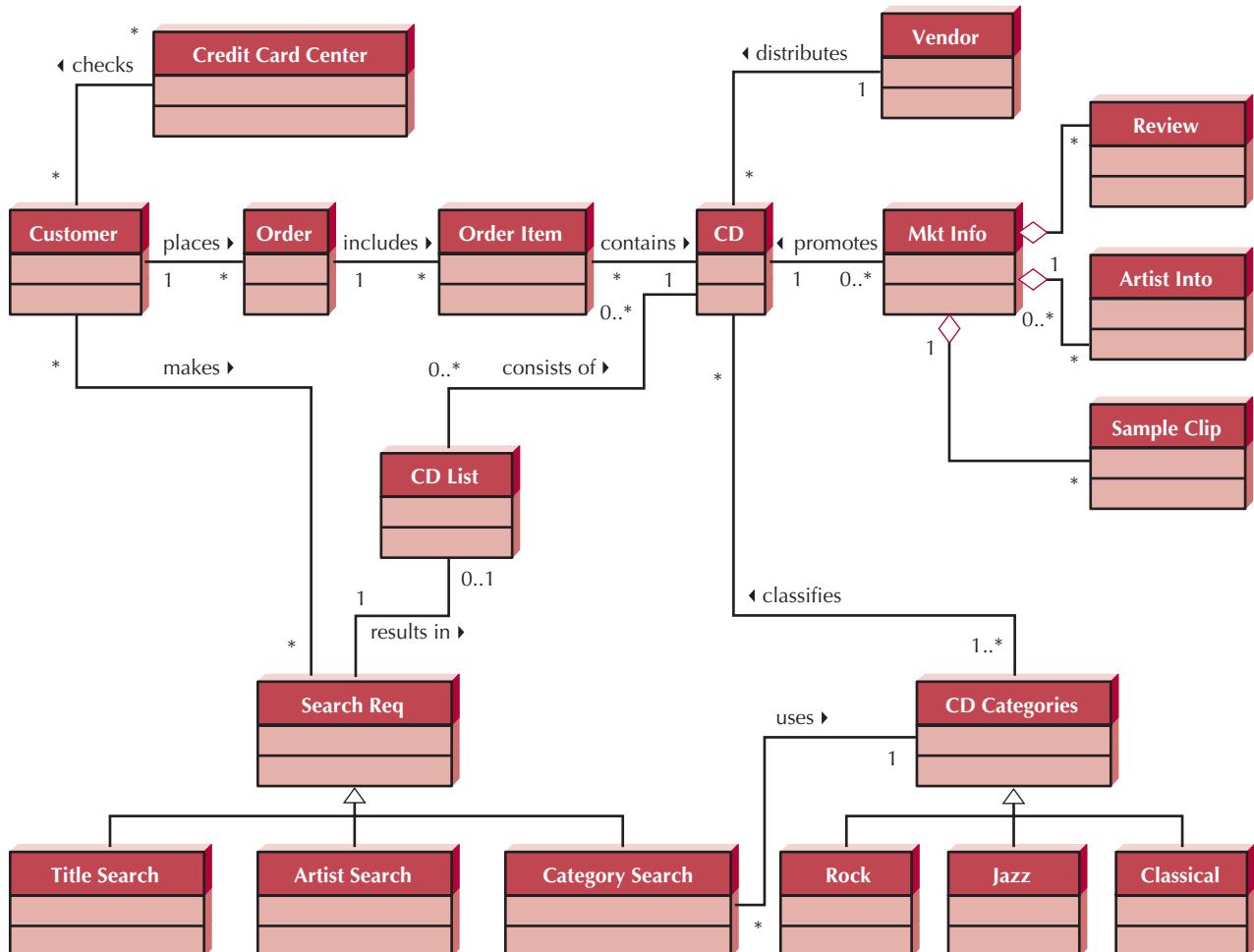
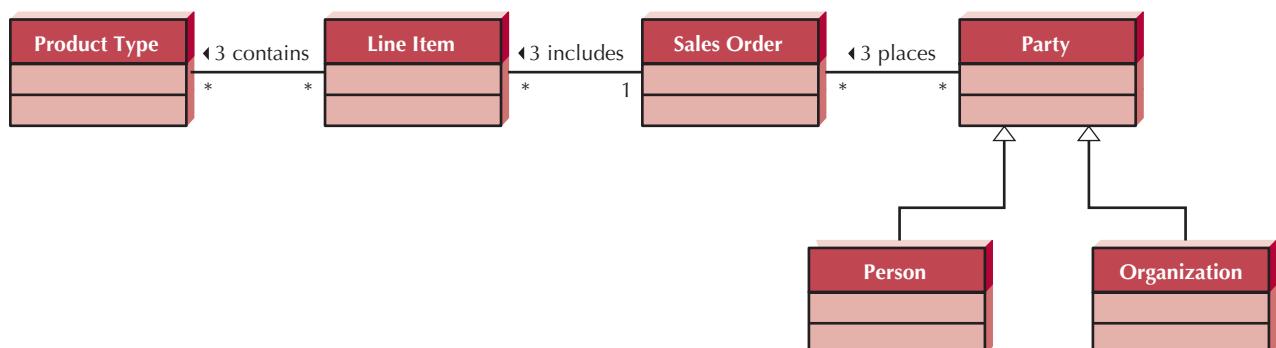


FIGURE 7-11 Preliminary CD Selections Internet Sales Class Diagram (Place Order Use-Case View)

FIGURE 7-12 Sales Order Contract Pattern¹⁰¹⁰ Adapted from David C. Hay, *Data Model Patterns: Conventions of Thought*.

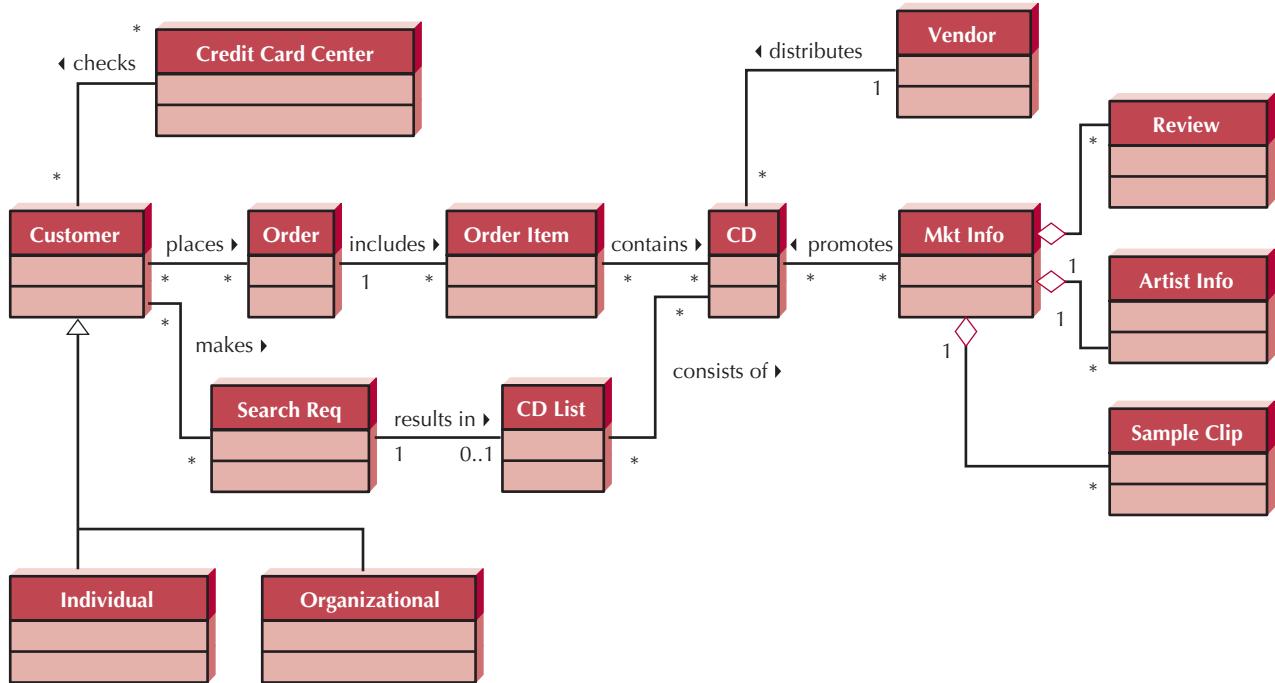


FIGURE 7-13 CD Selections Internet Sales System Class Diagram (Place Order Use-Case View)

**YOUR
TURN**

7-2 CD Selections Internet Sales System

In Your Turn 7-1, you completed the detailed use cases for the CD Selections Internet sales system. Using these detailed use cases, complete the structural model (CRC

cards and class diagram) for the remaining use cases in Figure 6-16.

**YOUR
TURN**

7-3 Campus Housing

In Your Turn 6-6, you created a set of the use cases from the campus housing service that helps students find apartments. Using the same use cases, create a structural model (CRC cards and class diagram). See if you can

identify at least one potential derived attribute, aggregation relationship, generalization relationship, and association relationship for the model.

SUMMARY

Structural Models

Structural models describe the underlying data structure of an object-oriented system. Where use-case models provide an external functional view of the evolving system (i.e., what the system does), structural models provide an internal static view of the evolving system (i.e., how the objects are organized in the system). At this point in the development of the system, the structural model only represents a logical model of the underlying problem domain. One of the primary purposes of the structural model is to create a vocabulary that allows the users and developers to effectively communicate about the problem under investigation. Structural models are comprised of classes, attributes, operations, and relationships. The three basic types of relationships that normally are depicted on structural models are aggregation, generalization, and association. Structural models typically are represented by CRC cards, class diagrams, and, in some cases, object diagrams.

CRC Cards

CRC cards model the classes, their responsibilities, and their collaborations. There are two different types of responsibilities: knowing and doing. Knowing responsibilities are associated mostly with attributes of instances of the class, while doing responsibilities are associated mostly with operations of instances of the class. Collaborations support the concepts of clients, servers, and contracts between objects. CRC cards capture all of the essential elements of the instances of a class. The front of the card allows the recording of the class's name, ID, type, description, list of associated use cases, responsibilities, and collaborators, while the back of the card contains the attributes and relationships.

Class and Object Diagrams

The class diagram is a graphical description of the information contained on the CRC cards. It shows the classes and the relationships between the classes. The visibility of the attributes and operations and the multiplicity of the relationships are additional information that the class diagram portrays, which is not included on the CRC cards. Also, at times a relationship itself contains information. In this case, an association class is created.

In real-world systems that can have over 100 classes, the class diagram can become overly complicated. To allow for the simplification of the diagram, a view mechanism can be used. A view restricts the amount of information portrayed on the diagram. Some useful views are: hiding all information about the class except for its name and relationships, showing only the classes that are associated with a particular use case, and limiting the relationships included to only one specific type (aggregation, generalization, or association).

When attempting to uncover additional information about the details of a class, it is useful to portray specific instances of a class instead of the classes themselves. An object diagram is used to depict a set of objects that represent an instantiation of all or part of a class diagram.

Creating CRC Cards and Class Diagrams

Creating a structural model is an iterative process. The process described is a seven-step process. The steps include textual analysis of use cases, brainstorming additional objects using common object lists, role-playing each use case using the CRC cards, creating class diagrams, and incorporating useful patterns.

KEY TERMS

A-kind-of	Decomposition	Protected
A-part-of	Derived attribute	Public
Abstract class	Doing responsibility	Query operation
Aggregation relationship	Generalization relationship	Responsibility
Anthropomorphism	Has-parts	Role-play
Assemblies	Incidents	Roles
Association relationship	Instances	Server
Association class	Instantiation	Specialization
Attribute	Interaction	Static model
Behavior	Knowing responsibility	Structural model
Class	Maximum number	Subclass
Class diagram	Minimum number	Substitutability
Client	Multiplicity	Superclass
Collaboration	Objects	SVDPI
Common object list	Object diagram	Tangible things
Conceptual model	Operation	Textual analysis
Concrete class	Package	Update operation
Constructor operation	Parts	View
Contract	Pattern	Visibility
CRC cards	Private	Wholes

QUESTIONS

- Describe to a businessperson the multiplicity of a relationship between two classes.
- Why are assumptions important to a structural model?
- What is an association class?
- Contrast the following sets of terms:
object; class; instance
property; method; attribute
superclass; subclass
concrete class; abstract class
- Give three examples of derived attributes that may exist on a class diagram. How would they be denoted on the class diagram?
- What are the different types of visibility? How would they be denoted on a class diagram?
- Draw the relationships that are described by the following business rules. Include the multiplicities for each relationship.

A patient must be assigned to only one doctor, and a doctor can have one or many patients.
An employee has one phone extension, and a unique phone extension is assigned to an employee.
A movie theater shows at least one movie, and a movie can be shown at up to four other movie theaters around town.
- A movie either has one star, two co-stars, or more than ten people starring together. A star must be in at least one movie.
- How do you designate the reading direction of a relationship on a class diagram?
- For what purpose is an association class used in a class diagram? Give an example of an association class that may be found in a class diagram that captures students and the courses they have taken.
- Give two examples of aggregation, generalization, and association relationships. How is each type of association depicted on a class diagram?
- Identify the following operations as constructor, query, or update. Which operations would not need to be shown in the class rectangle?

Calculate employee raise (raise percent)
Calculate sick days ()
Increment number of employee vacation days ()
Locate employee name ()
Place request for vacation (vacation day)
Find employee address ()
Insert employee ()
Change employee address ()
Insert spouse ()

EXERCISES

- A. Draw class diagrams for the following classes:
 Movie (title, producer, length, director, genre)
 Ticket (price, adult or child, showtime, movie)
 Patron (name, adult or child, age)
- B. Create an object diagram based on the class diagram you drew for Exercise A.
- C. Draw a class diagram for the following classes. Consider that the entities represent a system for a patient billing system. Include only the attributes that would be appropriate for this context.
 Patient (age, name, hobbies, blood type, occupation, insurance carrier, address, phone)
 Insurance carrier (name, number of patients on plan, address, contact name, phone)
 Doctor (specialty, provider identification number, golf handicap, age, phone, name)
- D. Create an object diagram based on the class diagram you drew for Exercise C.
- E. Draw the following relationships:
1. A patient must be assigned to only one patient, and a doctor can have many patients.
 2. An employee has one phone extension, and a unique phone extension is assigned to an employee.
 3. A movie theater shows many different movies, and the same movie can be shown at different movie theaters around town.
- F. Draw a class diagram for the following situations:
1. Whenever new patients are seen for the first time, they complete a patient information form that asks their name, address, phone number, and insurance carrier, which are stored in the patient information file. Patients can be signed up with only one carrier, but they must be signed up to be seen by the doctor. Each time a patient visits the doctor, an insurance claim is sent to the carrier for payment. The claim must contain information about the visit such as the date, purpose, and cost. It would be possible for a patient to submit two claims on the same day.
 2. The state of Georgia is interested in designing a system that will track its researchers. Information of interest includes researcher name, title, position; university name, location, enrollment; and research interests. Researchers are associated with one institution, and each researcher has several research interests.
 3. A department store has a bridal registry. This registry keeps information about the customer (usually the bride), the products that the store carries, and

the products each customer registers for. Customers typically register for a large number of products, and many customers register for the same products.

4. Jim Smith's dealership sells Fords, Hondas, and Toyotas. The dealership keeps information about each car manufacturer with whom it deals so that it can get in touch with them easily. The dealership also keeps information about the models of cars that it carries from each manufacturer. It keeps information such as list price, the price the dealership paid to obtain the model, and the model name and series (e.g., Honda Civic LX). The dealership also keeps information on all sales that it has made (for instance, it will record the buyer's name, the car bought, and the amount paid for the car). In order to contact the buyers in the future, contact information is also stored (e.g., address, phone number).
- G. Create object diagrams based on the class diagrams that you drew for Exercise F.
- H. Examine the class diagrams that you created for Exercise F. How would the models change (if at all) based on these new assumptions?
1. Two patients have the same first and last names.
 2. Researchers can be associated with more than one institution.
 3. The store would like to keep track of purchase items.
 4. Many buyers have purchased multiple cars from Jim over time because he is such a good dealer.
- I. Visit a Web site that allows customers to order a product over the Web (e.g., Amazon.com). Create a structural model (CRC cards and class diagram) that the site must need to support its business process. Include classes to show what they need information about. Be sure to include the attributes and operations to represent the type of information they use and create. Finally, draw relationships, making assumptions about how the classes are related.
- J. Create a structural model (CRC cards and class diagram) for Exercise C in Chapter 6.
- K. Create a structural model for Exercise E in Chapter 6.
- L. Create a structural model for Exercise H in Chapter 6.
- M. Create a structural model for Exercise J in Chapter 6.
- N. Create a structural model for Exercise L in Chapter 6.
- O. Create a structural model for Exercise N in Chapter 6.
- P. Create a structural model for Exercise P in Chapter 6.
- Q. Create a structural model for Exercise R in Chapter 6.
- R. Create a structural model for Exercise T in Chapter 6.

MINICASES

1. West Star Marinas is a chain of twelve marinas that offer lakeside service to boaters, service and repair of boats, motors, and marine equipment, and sales of boats, motors, and other marine accessories. The systems development project team at West Star Marinas has been hard at work on a project that eventually will link all the marina's facilities into one unified, networked system.

The project team has developed a use-case diagram of the current system. This model has been carefully checked. Last week, the team invited a number of system users to role-play the various use cases, and the use cases were refined to the users' satisfaction. Right now, the project manager feels confident that the As-Is system has been adequately represented in the use-case diagram.

The Director of Operations for West Star is the sponsor of this project. He sat in on the role playing of the use cases, and was very pleased by the thorough job the team had done in developing the model. He made it clear to you, the project manager, that he was anxious to see your team begin work on the use cases for the To-Be system. He was a little skeptical that it was necessary for your team to spend any time modeling the current system in the first place, but grudgingly admitted that the team really seemed to understand the business after going through that work.

The methodology you are following, however, specifies that the team should now turn its attention to developing the structural models for the As-Is system. When you stated this to the project sponsor, he seemed confused and a little irritated. "You are going to spend even more time looking at the current system? I thought you were done with that! Why is this necessary? I want to see some progress on the way things will work in the future!"

What is your response to the Director of Operations? Why do we perform structural modeling? Is there any benefit to developing a structural model of the current system at all? How does the use cases and use-case diagram help us develop the structural model?

2. Holiday Travel Vehicles sells new recreational vehicles and travel trailers. When new vehicles arrive at Holiday Travel Vehicles, a new vehicle record is created. Included in the new vehicle record is a vehicle serial number, name, model, year, manufacturer, and base cost.

When a customer arrives at Holiday Travel Vehicles, he/she works with a salesperson to negotiate a vehicle

purchase. When a purchase has been agreed to, a sales invoice is completed by the salesperson. The invoice summarizes the purchase, including full customer information, information on the trade-in vehicle (if any), the trade-in allowance, and information on the purchased vehicle. If the customer requests dealer-installed options, they will be listed on the invoice as well. The invoice also summarizes the final negotiated price, plus any applicable taxes and license fees. The transaction concludes with a customer signature on the sales invoice.

- a. Identify the classes described in the above scenario (you should find six). Create CRC cards for each of the classes.

Customers are assigned a customer ID when they make their first purchase from Holiday Travel Vehicles. Name, address, and phone number are recorded for the customer. The trade-in vehicle is described by a serial number, make, model, and year. Dealer-installed options are described by an option code, description, and price.

- b. Develop a list of attributes for each of the classes. Place the attributes onto the CRC cards.

Each invoice will list just one customer. A person does not become a customer until they purchase a vehicle. Over time, a customer may purchase a number of vehicles from Holiday Travel Vehicles.

Every invoice must be filled out by only one salesperson. A new salesperson may not have sold any vehicles, but experienced salespeople have probably sold many vehicles.

Each invoice only lists one new vehicle. If a new vehicle in inventory has not been sold, there will be no invoice for it. Once the vehicle sells, there will be just one invoice for it.

A customer may decide to have no options added to the vehicle, or may choose to add many options. An option may be listed on no invoices, or it may be listed on many invoices.

A customer may trade in no more than one vehicle on a purchase of a new vehicle. The trade-in vehicle may be sold to another customer, who later trades it in on another Holiday Travel Vehicle.

- c. Based on the above business rules in force at Holiday Travel Vehicles, and CRC cards draw a class diagram and document the relationships with the appropriate multiplicities. Remember to update the CRC cards.

CHAPTER 8

BEHAVIORAL MODELING

Behavioral models describe the internal dynamic aspects of an information system that supports the business processes in an organization. During the analysis phase, behavioral models describe what the internal logic of the processes is without specifying how the processes are to be implemented. Later, in the design and implementation phases, the detailed design of the operations contained in the object is fully specified. In this chapter, we describe three UML 2.0 diagrams that are used in behavioral modeling: sequence diagrams, communication diagrams, and behavioral state machines.

OBJECTIVES

- Understand the rules and style guidelines for sequence and communication diagrams and behavioral state machines.
- Understand the processes used to create sequence and communication diagrams and behavioral state machines.
- Be able to create sequence and communication diagrams and behavioral state machines.
- Understand the relationship between the behavioral models and the structural and functional models.

CHAPTER OUTLINE

Introduction	States, Events, Transitions, Actions, and Activities
Behavioral Models	Elements of a Behavioral State Machine
Interaction Diagrams	Building Behavioral State Machines
Objects, Operations, and Messages	Applying the Concepts at CD Selections
Sequence Diagrams	Summary
Communication Diagrams	
Behavioral State Machines	

INTRODUCTION

The previous two chapters discussed functional models and structural models. Systems analysts utilize functional models to describe the external behavioral view of an information system, while they use structural models to depict the static view of an information system. In this chapter, we discuss how analysts use behavioral models to represent the internal behavior or dynamic view of an information system.

There are two types of *behavioral models*. First, there are behavioral models that are used to represent the underlying details of a business process portrayed by a use case model. In UML, interaction diagrams (sequence and communication) are used for this type of behavioral model. Second, there is a behavioral model that is used to

represent the changes that occur in the underlying data. UML uses behavioral state machines for this.

During the analysis phase, analysts use behavioral models to capture a basic understanding of the dynamic aspects of the underlying business process. Traditionally, behavioral models have been used primarily during the design phase where analysts refine the behavioral models to include implementation details (see Chapter 9). For now, our focus is on *what* the dynamic view of the evolving system is and not on *how* the dynamic aspect of the system will be implemented.

In this chapter, we concentrate on creating behavioral models of the underlying business process. Using the interaction diagrams (sequence and communication diagrams) and behavioral state machines, it is possible to give a complete view of the dynamic aspects of the evolving business information system. We first describe behavioral models and their components. We then describe each of the diagrams, how they are created, and how they are related to the use case model and class diagram that are described in Chapters 6 and 7.

BEHAVIORAL MODELS

When an analyst is attempting to understand the underlying domain of a problem, he or she must consider both structural and behavioral aspects of the problem. Unlike other approaches to information systems development, object-oriented approaches attempt to view the underlying application domain in a holistic manner. By viewing the problem domain as a set of *use cases* that are supported by a set of collaborating objects, object-oriented approaches allow the analyst to minimize the semantic gap between the real-world set of objects and the evolving object-oriented model of the problem domain. However, as pointed out in the previous chapter, the real world tends to be messy, which makes perfectly modeling the problem domain practically impossible in software. This is because software must be neat and logical to work.

One of the primary purposes of behavioral models is to show how the underlying objects in the problem domain will collaborate to support each of the use cases. Whereas structural models represent the objects and the relationships between them, behavioral models depict the internal view of the business process that a use case describes. The process can be shown by the interaction that takes place between the objects that collaborate to support a use case through the use of interaction (sequence and communication) diagrams. It is also possible to show the effect that the set of use cases that make up the system has on the objects in the system through the use of behavioral state machines.

Creating behavioral models is an iterative process that not only iterates over the individual behavioral models (e.g., interaction (sequence and communication) diagrams and behavioral state machines) but also over the functional (see Chapter 6) and structural models (see Chapter 7). As the behavioral models are created, it is not unusual to make changes to the functional and structural models. In the next three sections, we describe interaction diagrams and behavioral state machines and when to use each.

INTERACTION DIAGRAMS

One of the primary differences between the class diagrams and the interaction diagrams, besides the obvious difference that one describes structure and the other behavior, is that the modeling focus on the class diagram is at the class level, while the interaction diagrams focus on the object level. In this section we review objects, operations, and messages and we cover the two different diagrams (sequence and communication) that can be used to model the interactions that take place between the objects in an information system.

Objects, Operations, and Messages

In Chapter 2, we defined an object as an instantiation of a *class*, that is, an actual person, place, event, or thing about which we want to capture information. If we were building an appointment system for a doctor's office, classes might include doctor, patient, and appointment. The specific patients like Jim Maloney, Mary Wilson, and Theresa Marks are considered objects—that is, *instances* of the patient class.

Each object has *attributes* that describe information about the object, such as a patient's name, birth date, address, and phone number. Each object also has *behaviors*. At this point in the development of the evolving system, the behaviors are described by *operations*. An operation is nothing more than an action that an object can perform. For example, an appointment object likely can schedule a new appointment, delete an appointment, and locate the next available appointment.

Each object also can send and receive messages. *Messages* are information sent to objects to tell an object to execute one of its behaviors. Essentially, a message is a function or procedure call from one object to another object. For example, if a patient is new to the doctor's office, the system will send an insert message to the application. The patient object will receive the instruction (the message) and do what it needs to do to go about inserting the new patient into the system (the behavior).

Sequence Diagrams

Sequence diagrams are one of two types of interaction diagrams. They illustrate the objects that participate in a use case and the messages that pass between them over time for *one* use case. A sequence diagram is a *dynamic model* that shows the explicit sequence of messages that are passed between objects in a defined interaction. Since sequence diagrams emphasize the time-based ordering of the activity that takes place among a set of objects, they are very helpful for understanding real-time specifications and complex use cases.

The sequence diagram can be a *generic sequence diagram* that shows all possible scenarios¹ for a use case, but usually each analyst develops a set of *instance sequence diagrams*, each of which depicts a single *scenario* within the use case. If you are interested in understanding the flow of control of a scenario by time, you should use a sequence diagram to depict this information. The diagrams are used throughout both the analysis and design phases. However, the design diagrams are very implementation-specific, often including database objects or specific GUI components as the classes. The following section presents the elements of a sequence diagram.

Elements of a Sequence Diagram Figure 8-1 shows an instance sequence diagram that depicts the objects and messages for the Make Appointment use case, which describes the process by which a patient creates a new appointment, cancels, or reschedules an appointment for the doctor's office appointment system. In this specific instance, the make appointment process is portrayed.

Actors and *objects* that participate in the sequence are placed across the top of the diagram using actor symbols from the use case diagram, and object symbols from the object diagram (see Figure 8-2). Notice that the actors and objects in Figure 8-1 are aPatient, aReceptionist, Patients, UnpaidBills, Appointments, and anAppt.² They are not placed in any particular order, although it is nice to organize them in some logical way, such as the order in which they participate in the sequence. For each of the objects, the name of the class that

¹ Remember that a scenario is a single executable path through a use case.

² In some versions of the sequence diagram, object symbols are used as surrogates for the actors. However, for purposes of clarity, we recommend using actor symbols for actors instead.

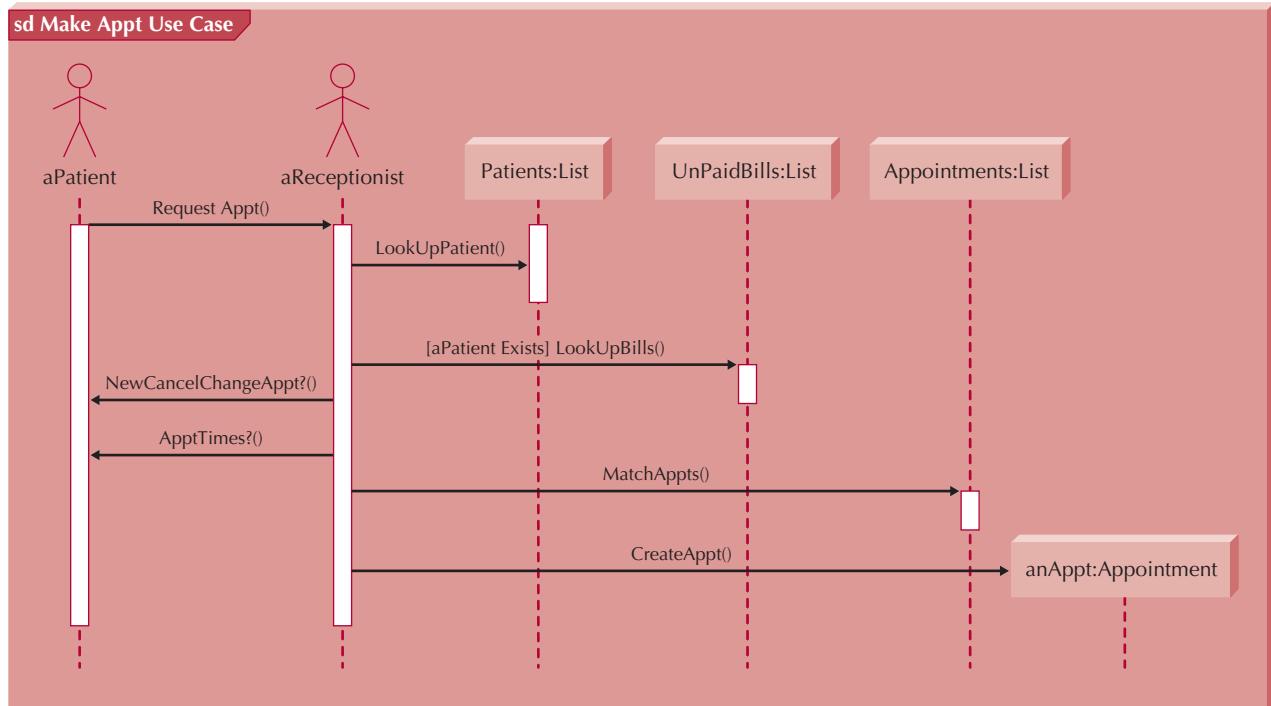


FIGURE 8-1 Example Sequence Diagram

they are an instance of is given after the object's name (e.g., Patients:List means that Patients is an instance of the List class that contains individual patient objects).

A dotted line runs vertically below each actor and object to denote the *lifeline* of the actors/objects over time (see Figure 8-1).³ Sometimes an object creates a *temporary object*, and in this case an X is placed at the end of the lifeline at the point the object is destroyed (not shown). For example, think about a shopping cart object for a Web commerce application. The shopping cart is used for temporarily capturing line items for an order, but once the order is confirmed, the shopping cart is no longer needed. In this case, an X would be located at the point at which the shopping cart object is destroyed. When objects continue to exist in the system after they are used in the sequence diagram, then the lifeline continues to the bottom of the diagram (this is the case with all of the objects in Figure 8-1).

A thin rectangular box, called the *execution occurrence*, is overlaid onto the lifeline to show when the classes are sending and receiving messages (see Figure 8-2). A *message* is a communication between objects that conveys information with the expectation that activity will ensue. There are many different types of messages that can be portrayed on a sequence diagram. However, in the case of using sequence diagrams to model use cases, two types of messages are typically used: Operation Call and Return. *Operation call messages* passed between classes are shown using solid lines connecting two objects with an arrow on the line showing which way the message is being passed. Argument values for the message are placed in parentheses next to the message's name. The order of messages goes from the top to the bottom of the page, so messages located higher on the diagram represent messages

³ Technically speaking, in UML 2.0 the lifeline actually refers to both the object (actor) and the dashed line drawn vertically underneath the object (actor). However, we prefer to use the older terminology because it is more descriptive of what is actually being represented.

An Actor:	<ul style="list-style-type: none"> Is a person or system that derives benefit from and is external to the system Participates in a sequence by sending and/or receiving messages Are placed across the top of the diagram Is depicted as either a stick figure (default) or if a non-human actor is involved, as a rectangle with <> in it (alternative) 	
An Object:	<ul style="list-style-type: none"> Participates in a sequence by sending and/or receiving messages Are placed across the top of the diagram 	
A Lifeline:	<ul style="list-style-type: none"> Denotes the life of an object during a sequence Contains an "X" at the point at which the class no longer interacts 	
An Execution Occurrence:	<ul style="list-style-type: none"> Is a long narrow rectangle placed atop a lifeline Denotes when an object is sending or receiving messages 	
A Message:	<ul style="list-style-type: none"> Conveys information from one object to another one An operation call is labeled with the message being sent and a solid arrow, while a return is labeled with the value being returned and shown as a dashed arrow 	
Object Destruction:	<ul style="list-style-type: none"> An X is placed at the end of an object's lifeline to show that it is going out of existence 	
A Frame:	<ul style="list-style-type: none"> Indicates the context of the sequence diagram 	

FIGURE 8-2 Sequence Diagram Syntax

that occur earlier on in the sequence, versus the lower messages that occur later. A *return message* is depicted as a dashed line with an arrow on the end of the line portraying the direction of the return. The information being returned is used to label the arrow. However, since adding return messages tends to clutter the diagram, unless the return messages add a lot of information to the diagram, they should be omitted. In Figure 8-1, “LookUpPatient” is a message sent from the actor aReceptionist to the object Patients, which is a container for the current patients to determine whether the aPatient actor is a current patient.

At times a message is sent only if a *condition* is met. In those cases, the condition is placed between a set of [] (e.g., [aPatient Exists] LookupBills()). The condition is placed in front of the message name. However, when using a sequence diagram to model a specific

scenario, conditions typically are not shown on any single sequence diagram. Instead, conditions are implied only through the existence of different sequence diagrams. There are times that a message is repeated. This is designated with an * in front of the message name (e.g., * Request CD). An object can send a message to itself. This is known as *self-delegation*.

Sometimes, an object will create another object. This is shown by the message being sent directly to an object instead of its lifeline. In Figure 8-1, the actor aReceptionist creates an object anAppt.

Building a Sequence Diagram In this section we describe a six-step process used to create a sequence diagram⁴ (see Figure 8-3). The first step in the process is to determine the context of the sequence diagram. The context of the diagram can be a system, a use case, a scenario of a use case, or an operation of a class. The context of the diagram is depicted as a labeled *frame* around the diagram (see Figures 8-1 and 8-2). Most commonly, it is one use case scenario.

The second step is to identify the objects that participate in the sequence being modeled—that is, the objects that interact with each other during the use case scenario. The objects are identified during the development of the structural model (see Chapter 7). These are the classes on which the objects of the sequence diagram for this scenario will be based. However, during this process, it is likely that new classes, and hence new objects, will be uncovered. Don't worry too much about identifying all the objects perfectly; remember that the behavioral modeling process is iterative. Usually, the sequence diagrams are revised multiple times during the structural and behavioral modeling processes.

The third step is to set the lifeline for each object. To do this, you need to draw a vertical dotted line below each class to represent the class's existence during the sequence. An X should be placed below the object at the point on the lifeline where the object goes out of existence.

The fourth step is to add the messages to the diagram. This is done by drawing arrows to represent the messages being passed from object to object, with the arrow pointing in the message's transmission direction. The arrows should be placed in order from the first message (at the top) to the last (at the bottom) to show time sequence. Any parameters passed along with the messages should be placed in parentheses next to the message's name. If a message is expected to be returned as a response to a message, then the return message is not explicitly shown on the diagram.

The fifth step is to place the execution occurrence on each object's lifeline by drawing a narrow rectangle box over top of the lifelines to represent when the classes are sending and receiving messages.

The sixth and final step is to validate the sequence diagram. The purpose of this step is to guarantee that the sequence diagram completely represents the underlying process(es). This is done by guaranteeing that the diagram depicts all of the steps in the process.

1. Set the context.
2. Identify which objects will participate.
3. Set the lifeline for each object.
4. Layout the messages from the top to the bottom of the diagram based on the order in which they are sent.
5. Add the execution occurrence to each object's lifeline.
6. Validate the sequence diagram.

FIGURE 8-3
Steps for Building
Sequence Diagrams

⁴ The approach described in this section are adapted from Grady Booch, James Rumbaugh, Ivar Jacobson, *The Unified Modeling Language User Guide* (Reading, MA: Addison-Wesley, 1999).

Applying the Concepts at CD Selections The best way to learn how to build a sequence diagram is to draw one. In this section we apply the steps just described to the CD Selections case used in the previous chapters.

The first step in the process is to determine the context of the sequence diagram. We will use a scenario⁵ from the Place Order use case that was created in Chapter 6 and illustrated in Figure 6-15. (Refer to the original use case for the details.) Figure 8-4 lists the Normal Flow of Events that contains the scenario that this sequence diagram will need to describe.

The second step is to identify the objects that participate in the sequence being modeled. The classes associated with the Place Order use case are shown in Figure 7-13. For example, the classes used for the Place Order use case include Customer, CD, Marketing Information, Credit Card Center, Order, Order Item, Vendor, Search Request, CD List, Review, Artist Information, Sample Clip, Individual, and Organizational.

During the role playing of the *CRC Cards*, one of the analysts assigned to the CD Selections Internet System Development Team asked whether a Shopping Cart class should be included. She stated that every Internet sales site she had been to had a shopping cart that was used to build an order. However, the instance of the Shopping Cart class only existed until either an order was placed or the shopping cart was emptied. Based on this obvious oversight, both the Place Order use case and the class diagram will have to be modified. Another analyst pointed out that the CDs themselves were going to have to be associated with some type of searchable storage. Otherwise, it would be impossible to find the CD in which the customer was interested. However, it was decided that the CD List class would suffice for both the searchable storage and a temporary instance that is created as a result of a search request. This again is a fairly typical situation in object-oriented development. The more the development team learns about the requirements, the more the models (functional, structural, and behavioral) will evolve. The important thing is to remember that an object-oriented development process is iterative over all of the models.

The objects that will be needed to describe this scenario are instances of the Search Request, CD List, CD, Marketing Material, Customer, Review, Artist Information, Sample Clip, and Shopping Cart classes. We also have an actor (Customer) that interacts with the scenario. To complete this step, you should lay out the objects on the sequence diagram by drawing them, from left to right, across the diagram.

The third step is to set the lifeline for each object. To do this, you should draw a vertical dotted line below each of the objects (aSR, aCDL, CDs, aCD, MI, ar, AI, SC, and anOrder) and the actor (aCustomer). Also, an X should be placed at the bottom of the lifelines for aCDL and aSC since they go away at the end of this process.

FIGURE 8-4
Normal Flow of Events
of the Places Order
Use Case

Normal Flow of Events:

1. Customer submits a search request to the system.
2. The System provides the Customer a list of recommended CDs.
3. The Customer chooses one of the CDs to find out additional information.
4. The System provides the Customer with basic information and reviews on the CD.
5. The Customer calls the Maintain Order use case.
6. The Customer iterates over 3 through 5 until done shopping.
7. The Customer executes the Checkout use case.
8. The Customer leaves the Web site.

⁵ Remember, as stated previously, a scenario is a single executable path through a use case.

The fourth step is to add the messages to the diagram. Examine the steps in Figure 8-4 and see if you can determine the way in which the messages should be added to the sequence diagram. Figure 8-5 shows our results. Notice how we did not include messages back to Customer in response to “create SR” and “add CD.” In these cases, it is assumed that the customer will receive response messages about the requested CD and inserted CD, respectively.

The fifth step is to add the execution occurrence to each object’s and actor’s lifeline. This is done by drawing a narrow rectangle box over top of the lifelines to represent when the objects (actors) are sending and receiving messages (e.g., in Figure 8-5 aCustomer is active during the entire process, while aSR is only active at the beginning of the process [the top of the diagram]).

Finally, the CD Selections team validated the diagram by ensuring that the diagram accurately and completely represents the underlying process associated with the Place Order use case. Figure 8-5 portrays their completed sequence diagram.

Communication Diagrams

Communication diagrams, like sequence diagrams, essentially provide a view of the dynamic aspects of an object-oriented system. A communication diagram is essentially an object diagram that shows message passing relationships instead of aggregation or generalization associations. Communication diagrams are very useful to show process patterns (i.e., patterns of activity that occur over a set of collaborating classes).

Communication diagrams are equivalent to sequence diagrams, but they emphasize the flow of messages through a set of objects, while the sequence diagrams focus on the time ordering of the messages being passed. Therefore, if you are interested in understanding the flow of control over a set of collaborating objects, you should use a communication diagram. If you are interested in the time ordering of the messages, you should use a sequence diagram. In some cases, you will want to use both to more fully understand the dynamic activity of the system.

Elements of a Communication Diagram A communication diagram for the Make Appointment use case is portrayed in Figure 8-6. Like the sequence diagram in Figure 8-1, the appointment creation process is portrayed.

Actors and objects that collaborate to execute the use case are placed on the communication diagram in a manner to emphasize the message passing that takes place between them. Notice that the actors and objects in Figure 8-6 are the same ones in Figure 8-1: aPatient, aReceptionist, Patients, UnpaidBills, Appointments, and anAppt.⁶ Again, like the sequence diagram, for each of the objects, the name of the class that they

YOUR TURN

8-1 Drawing a Sequence Diagram

In Your Turn 6-6, you were asked to create a set of use cases and a use-case diagram for the campus housing service that helps students find apartments. In Your Turn 7-3, you were asked to create a structural model (CRC Cards

and class diagram) for those use cases. Select one of the use cases from the use case diagram and create a sequence diagram that represents the interaction among classes in the use case.

⁶ In some versions of the communication diagram, object symbols are used as surrogates for the actors. However, again we recommend using actor symbols for actors instead.

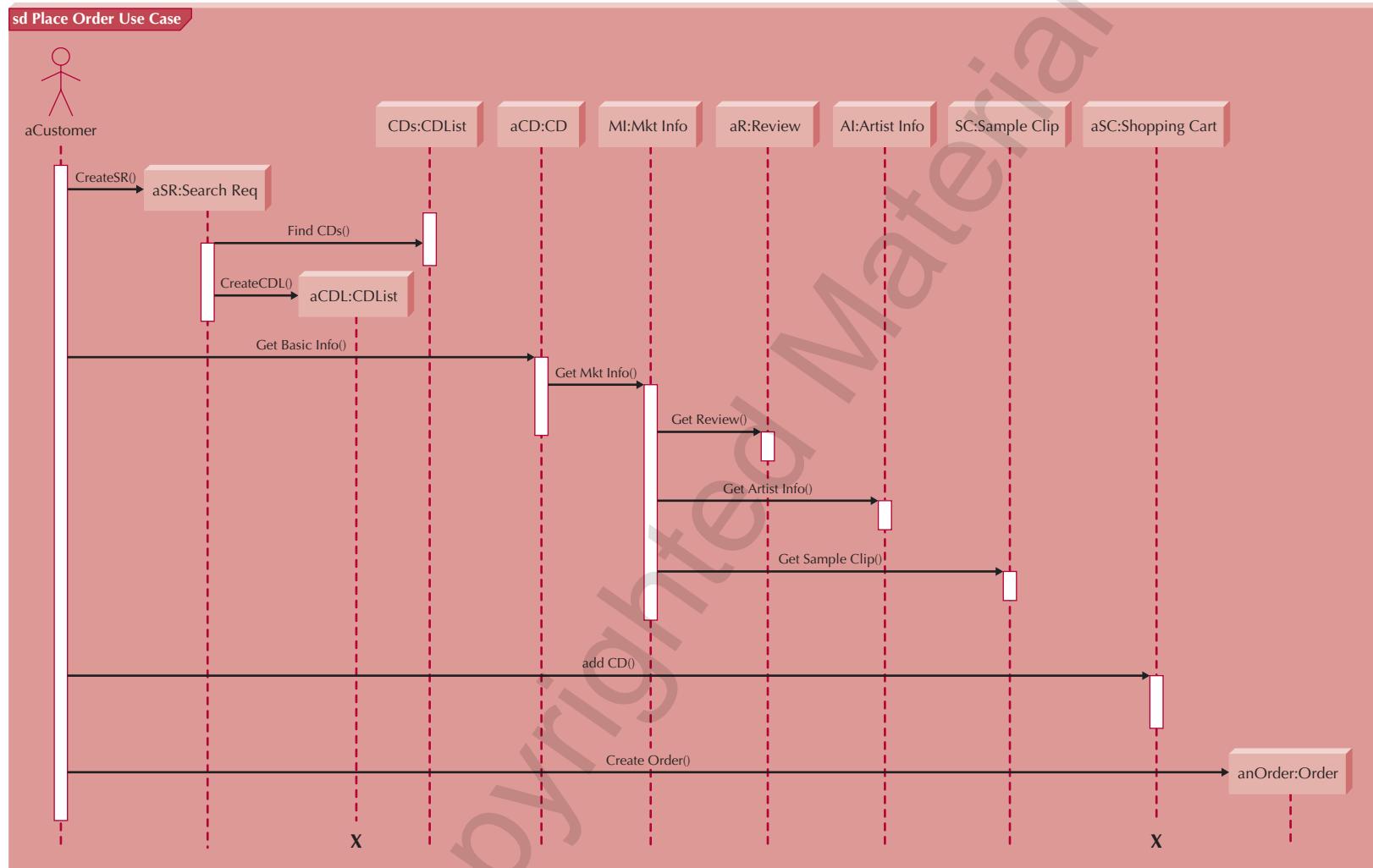


FIGURE 8-5 Sequence Diagram for the Places Order Use Case

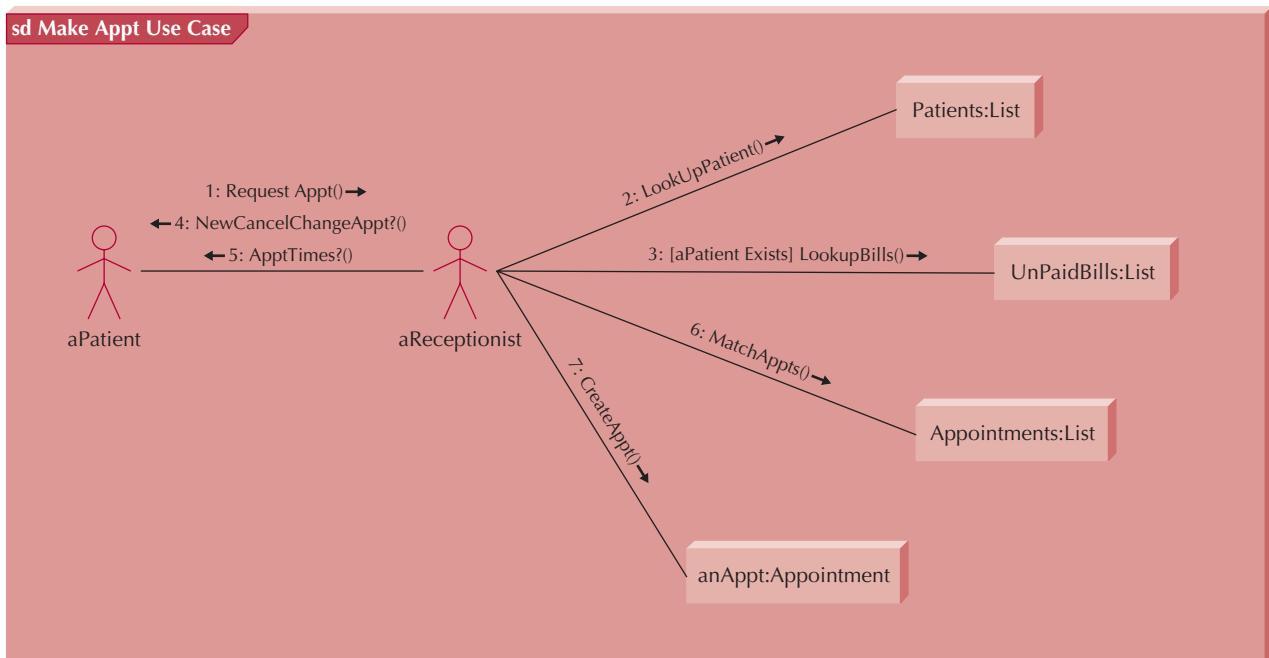


FIGURE 8-6 Example Communication Diagram

are an instance of is given after the object's name (e.g., Patients : List). (See Figure 8-7 for the Communication Diagram Syntax.) Unlike the sequence diagram, the communication diagram does not have a means to explicitly show an object being deleted or created. It is assumed that when a delete, destroy, or remove message is sent to an object, it will go out of existence, and a create or new message will cause a new object to come into existence. Another difference between the two interaction diagrams is that the communication diagram never shows returns from message sends, while the sequence diagram can optionally show them.

An *association* is shown between actors and objects with an undirected line. For example, there is an association shown between the aPatient and aReceptionist actors. Messages are shown as labels on the associations. Included with the labels are lines with arrows showing the direction of the message being sent. For example, in Figure 8-6, the aPatient actor sends the RequestAppt() message to the aReceptionist actor and the aReceptionist actor sends the NewCancelChangeAppt? and the ApptTimes? messages to the aPatient actor. The sequence of the message sends is designated with a sequence number. For example, in Figure 8-6, the RequestAppt() message is the first message sent, while the NewCancelChangeAppt? and the ApptTimes? messages are the fourth and fifth message sent, respectively.

Like the sequence diagram, the communication diagram can represent conditional messages. For example in Figure 8-6, the LookupBills() message is only sent if the [aPatient exists] condition is met. If a message is repeatedly sent, an * is placed after the sequence number. Finally, an association that loops onto an object shows self-delegation. The message is shown as the label of the association.

When a communication diagram is fully populated with all of the objects, it can become very complex and difficult to understand. When this occurs, it is necessary to simplify the diagram. One approach to simplify a communication diagram, like use case diagrams (see Chapter 6) and class diagrams (see Chapter 7), is through the use of *packages*.

<p>An Actor:</p> <ul style="list-style-type: none"> ■ Is a person or system that derives benefit from and is external to the system ■ Participates in a collaboration by sending and/or receiving messages 	 <p>anActor</p> <div style="border: 1px solid black; padding: 5px; background-color: #e0c0c0;"><<actor>> Actor/Role</div>
<p>An Object:</p> <ul style="list-style-type: none"> ■ Participates in a collaboration by sending and/or receiving messages ■ Are placed across the top of the diagram 	<div style="border: 1px solid black; padding: 5px; background-color: #e0c0c0;">anObject : aClass</div>
<p>An Association:</p> <ul style="list-style-type: none"> ■ Shows an association between actors and/or objects ■ Messages are sent over associations 	<hr/>
<p>A Message:</p> <ul style="list-style-type: none"> ■ Conveys information from one object to another one ■ Direction is shown using an arrowhead ■ Sequence is shown by a sequence number 	<div style="text-align: right;">1: a Message() →</div>
<p>A Frame:</p> <ul style="list-style-type: none"> ■ Indicates the context of the communication diagram 	<div style="border: 1px solid black; padding: 5px; background-color: #e0c0c0;">Context</div>

FIGURE 8-7 Communication Diagram Syntax

(i.e., logical groups of classes). In the case of communication diagrams, its objects are grouped together based on the messages sent to and received from other objects.⁷

Building a Communication Diagram⁸ Remember that a communication diagram is basically an object diagram that shows message-passing relationships instead of aggregation or generalization associations. In this section, we describe a five-step process used to build a communication diagram (see Figure 8-8). The first step in the process is to determine the context of the communication diagram. Like a sequence diagram, the context of the diagram can be a system, a use case, a scenario of a use case, or an operation of a class. The context of the diagram is depicted as a labeled frame around the diagram (see Figures 8-6 and 8-7).

The second step is to identify the objects (actors) and the associations that link the objects (actors) that participate in the collaboration together. Remember, the objects that participate in the *collaboration* are instances of the classes identified during the

⁷ For those familiar with structured analysis and design, packages serve a similar purpose as the leveling and balancing processes used in data-flow diagramming. Packages and Package diagrams are described in Chapter 9.

⁸ The approach described in this section are adapted from Booch, Rumbaugh, and Jacobson, *The Unified Modeling Language User Guide*.

FIGURE 8-8
Steps for Building
Communication
Diagrams

1. Set the context.
2. Identify which objects (actors) and the associations between the objects participate in the collaboration.
3. Layout the communication diagram.
4. Add messages.
5. Validate the communication diagram.

development of the structural model (see Chapter 7). Like the sequence diagramming process, it is likely that additional objects, and hence classes, will be discovered. Again, this is normal since the underlying development process is iterative and incremental.

One useful technique to identify potential collaborations is *CRUD analysis*.⁹ CRUD analysis uses a *CRUD matrix* in which each interaction among objects is labeled with a letter for the type of interaction: C for Create, R for Read or Reference, U for update, and D for Delete. In an object-oriented approach, a class-by-class matrix is used. Each cell in the matrix represents the interaction between instances of the classes. For example in Figure 8-5, an instance of SearchReq created an instance of CDList. As such, assuming a Row:Column ordering, a “C” would be placed in the cell SearchReq:CDList. Also, in Figure 8-5, an instance of CD references an instance of MktInfo. In this case, an “R” would be placed in the CD:Mkt-Info cell. Figure 8-9 shows the CRUD matrix based on the Place Order use case.

Once a CRUD matrix is completed for the entire system, the matrix can be scanned quickly to ensure that each and every class can be instantiated. Furthermore, each type of interaction can be validated for each class. For example, if a class only represents temporary objects, then the column in the matrix should have a D in it somewhere. Otherwise, the instances of the class will never be deleted. Since a data warehouse contains historical data, objects that are

	Customer	SearchReq	CDList	CD	Mkt Info	Review	Artist Info	Sample Clip	Shopping Cart	Order
Customer		R							U	C
SearchReq			CR							
CDList										
CD					R					
Mkt Info						U	U	U		
Review										
Artist Info										
Sample Clip										
Shopping Cart										
Order										

FIGURE 8-9 CRUD Matrix for the Place Order Use Case

⁹ CRUD analysis has typically been associated with structured analysis and design (see Alan Dennis and Barbara Haley Wixom, *Systems Analysis Design*, 2nd Ed., New York: Wiley, 2000) and information engineering (see James Martin, *Information Engineering, Book II Planning and Analysis*, Englewood Cliffs, NJ: Prentice Hall, 1990). In our case, we have simply adapted it to object-oriented systems development.

to be stored in one should not have any U or D entries in their associated columns. As such, CRUD analysis can be used as a way to partially validate the interactions among the objects in an object-oriented system. Finally, the more interactions among a set of classes, the more likely they should be clustered together in a collaboration. However, the number and type of interactions are only an estimate at this point in the development of the system. As such, care should be taken when using this technique to cluster classes together to identify collaborations.

The third step is to lay out the objects (actors) and their associations on the communication diagram by placing them together based on the associations that they have with the other objects in the collaboration. By focusing on the associations between the objects (actors) and minimizing the number of associations that cross over one another, we can increase the understandability of the diagram.

The fourth step is to add the messages to the associations between the objects. We do this by adding the name of the message(s) to the association link between the objects and an arrow showing the direction of the message being sent. Furthermore, each message has a sequence number associated with it to portray the time-based ordering of the message.¹⁰

The fifth and final step is to validate the communication diagram. The purpose of this step is to guarantee that the communication diagram faithfully portrays the underlying process(es). This is done by ensuring that all steps in the process are depicted on the diagram.

Applying the Concepts at CD Selections Like creating sequence diagrams, the best way to learn how to create a communication diagram is to draw one. The first step in the process is to determine the context of the communication diagram. We will use the same scenario of the Place Order use case that we used previously with the description of creating sequence diagrams (see Figure 8-10).

By executing the second step, the CD Selections team identifies the objects and the associations that link the objects together. Since we are using the same scenario as we did in the sequence diagram previously described, they need to include the same objects: instances of the Search Request, CD List, CD, Marketing Material, Customer, Review, Artist Information, Sample Clip, and Shopping Cart classes. And again, we have an actor (Customer) that interacts with the scenario. Furthermore, the team identified the associations between the objects (e.g., the instances of CD are associated with instances of Mkt Info, which, in turn, are associated with instances of Review, Artist Info, and Sample Clip).

During the third step, the team placed the objects on the diagram based on the associations that they have with the other objects in the collaboration. This was done to increase the readability, and hence, the understandability of the diagram (see Figure 8-10).

During the fourth step, the CD Selections team added the messages to the associations between the objects. For example, in Figure 8-10, the Create SR() message is the first message sent and the FindCDs() message is the second message sent. The aCustomer actor sends the Create SR() message to the aSR object, and the aSR object sends the FindCDs() message to the CDs object.

Finally, the CD Selections team executed the fifth and final step: validating the diagram. They accomplished this by ensuring that the underlying process associated with the Place Order use case was accurately and completely represented by the diagram. See Figure 8-10 for the completed communication diagram for this particular scenario of the Place Order use case.

At this point in time, you should review Figures 8-5 and 8-10. These two different diagrams are simply used to provide two different views of the same scenario. However, as you can see, it is easier to portray the time ordering of the messages in the sequence diagram

¹⁰ However, remember the sequence diagram portrays the time-based ordering of the messages in a top-down manner. As such, if your focus is on the time-based ordering of the messages, we recommend that you also use the sequence diagram.

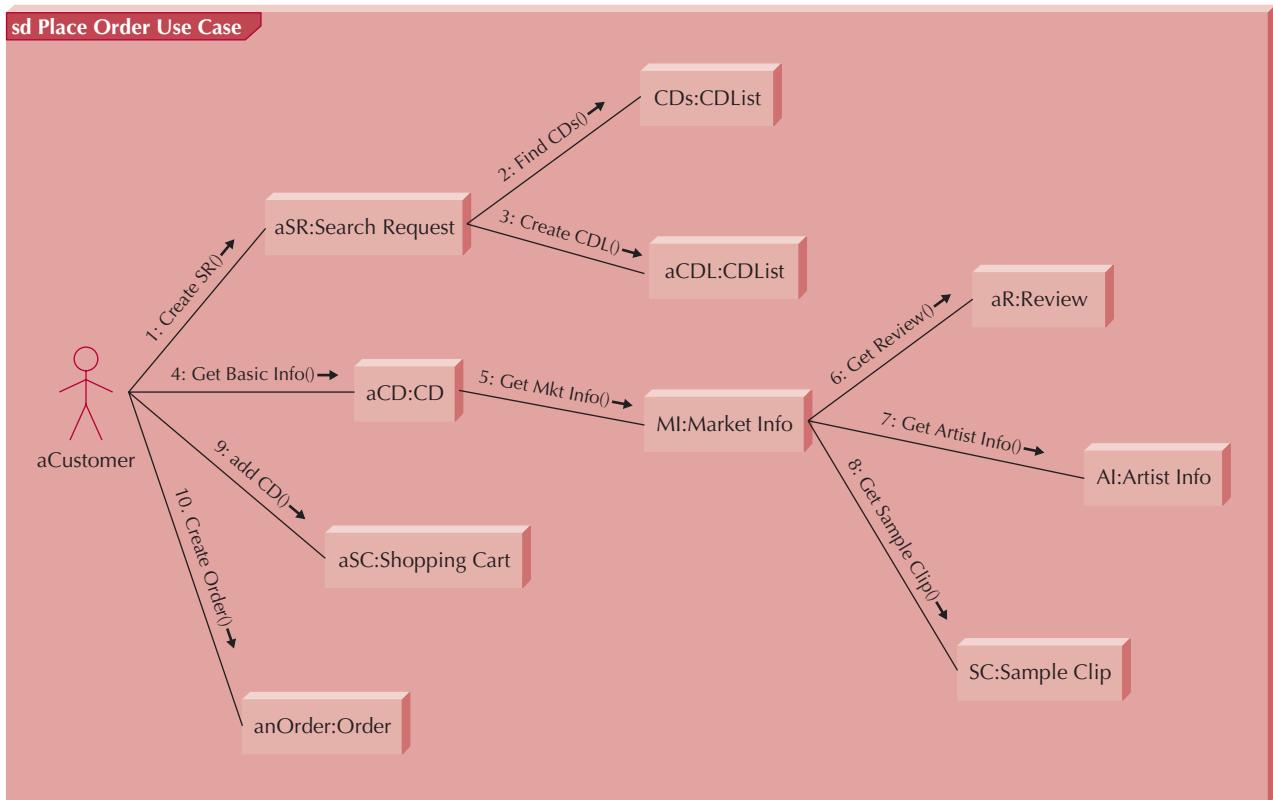


FIGURE 8-10 Communication Diagram for the Place Order Use Case

than it is in the communication diagram, and it is easier to represent how the objects interact to implement the use case using the communication diagram.

BEHAVIORAL STATE MACHINES

Some of the classes in *class diagrams* represent a set of objects that are quite dynamic in that they pass through a variety of states over the course of their existence. For example, a patient can change over time from being “new” to “current,” “former,” and so on, based on his or her status with the doctor’s office. A *behavioral state machine* is a *dynamic model* that shows the different states that a single object passes through during its life in response to events, along with its responses and actions. Typically, behavioral state machines are not used for all

YOUR TURN

8-2 Drawing a Communication Diagram

In Your Turn 8-1, you were asked to create a sequence diagram for the campus housing service. Draw a commu-

nication diagram for the same situation.

objects, but just to further define complex objects to help simplify the design of algorithms for their methods. CRUD analysis is one technique (see Communication Diagram section) that can be used to identify complex objects. The more (C)reate, (U)pdate or (D)elete entries in the column associated with a class, the more likely the instances of the class will have a complex life cycle. As such, these objects are candidates for state modeling with a behavioral state machine. The behavioral state machine shows the different states of the object and what events cause the object to change from one state to another. In comparison to the interaction diagrams, behavioral state machines should be used if you are interested in understanding the dynamic aspects of a single class and how its instances evolve over time, and not with how a particular use case scenario is executed over a set of classes. In this section, we describe states, events, transitions, actions, and activities and the use of the behavioral state machine to model the state changes through which complex objects pass.

States, Events, Transitions, Actions, and Activities

The *state* of an object is defined by the value of its attributes and its relationships with other objects at a particular point in time. For example, a patient might have a state of “new” or “current” or “former.” The attributes or properties of an object affect the state that it is in; however, not all attributes or attribute changes will make a difference. For example, think about a patient’s address. Those attributes make very little difference as to changes in a patient’s state. However, if states were based on a patient’s geographic location (e.g., in-town patients were treated differently than out-of-town patients), changes to the patient’s address would influence state changes.

An *event* is something that takes place at a certain point in time and changes a value(s) that describes an object, which, in turn, changes the object’s state. It can be a designated condition becoming true, the receipt of the call for a method by an object, or the passage of a designated period of time. The state of the object determines exactly what the response will be.

A *transition* is a relationship that represents the movement of an object from one state to another state. Some transitions will have a guard condition. A *guard condition* is a Boolean expression that includes attribute values, which allows a transition to occur only if the condition is true. An object typically moves from one state to another based on the outcome of an action that is triggered by an event. An *action* is an atomic, nondecomposable process than cannot be interrupted. From a practical perspective, actions take zero time, and they are associated with a transition. In contrast, an *activity* is a nonatomic, decomposable process that can be interrupted. Activities take a long period of time to complete, they can be started and stopped by an action, and they are associated with states.

Elements of a Behavioral State Machine

Figure 8-11 presents an example of a behavioral state machine representing the patient class in the context of a hospital environment. From this diagram we can tell that a patient enters a hospital and is admitted after checking in. If a doctor finds the patient to be healthy, he or she is released, and is no longer considered a patient after two weeks elapses. If a patient is found to be unhealthy, he or she remains under observation until the diagnosis changes.

A *state* is a set of values that describes an object at a specific point in time, and it represents a point in an object’s life in which it satisfies some condition, performs some action, or waits for something to happen (see Figure 8-12). In Figure 8-11 states include entering, admitted, released, and under observation. A state is depicted by a *state symbol*, which is a rectangle with rounded corners with a descriptive label that communicates a particular state. There are two exceptions. An *initial state* is shown using a small solid filled-in circle, and an object’s *final state* is shown as a circle surrounding a small solid filled-in circle. These exceptions depict when an object begins and ceases to exist, respectively.

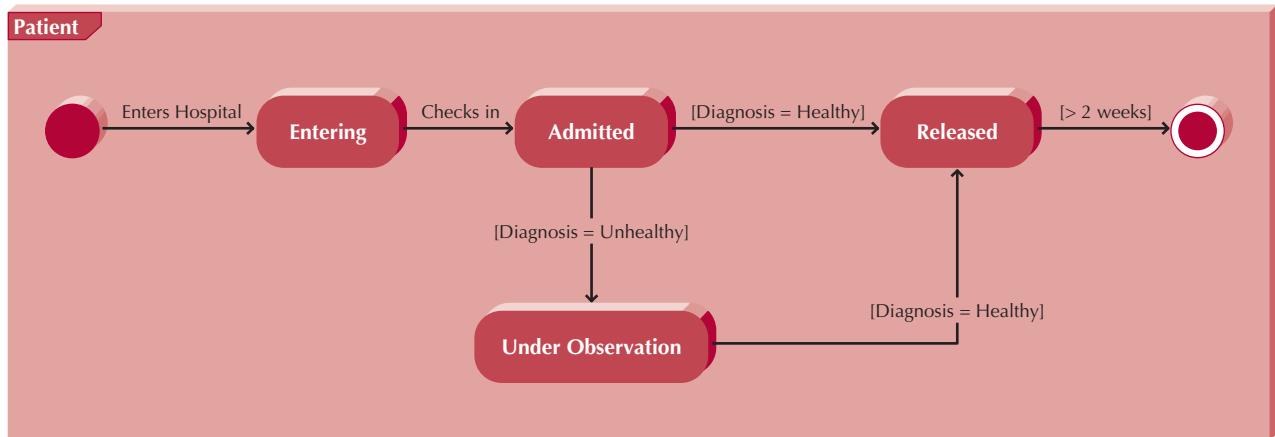


FIGURE 8-11 Example Behavioral State Machine Diagram

Arrows are used to connect the state symbols, representing the *transitions* between states. Each arrow is labeled with the appropriate event name and any parameters or conditions that may apply. For example, the two transitions from admitted to released and under observation contain guard conditions. As in the other behavioral diagrams, in many cases it is useful to explicitly show the context of the behavioral state machine using a frame.

Building a Behavioral State Machine

Behavioral state machines are drawn to depict a single class from a class diagram. Typically the classes are very dynamic and complex, requiring a good understanding of their states over time and events triggering changes. You should examine your class diagram to identify which classes will need to undergo a complex series of state changes and draw a diagram for each of them. In this section, we describe a five-step process used to build a behavioral state machine¹¹ (see Figure 8-13). Like the other behavioral models, the first step in the process is to determine the context of the behavioral state machine, which is shown in the label of the frame of the diagram. The context of a behavioral state machine is usually a class. However, it also could be a set of classes, a subsystem, or an entire system.

The second step is to identify the various states that an object will have over its lifetime. This includes establishing the boundaries of the existence of an object by identifying the initial and final states of an object. We also must identify the stable states of an object. The information necessary to perform this is gleaned from reading the use case descriptions, talking with users, and relying on the requirements gathering techniques that you learned about in Chapter 5. An easy way to identify the states of an object is to write the steps of what happens to an object over time, from start to finish, similar to how you would create the Normal Flow of Events section of a use case description.

The third step is to determine the sequence of the states that an object will pass through during its lifetime. Using this sequence, the states are placed onto the behavioral state machine, in a left-to-right order.

The fourth step is to identify the events, actions, and guard conditions associated with the transitions between the states of the object. The events are the *triggers* that cause an object to move from one state to the next state. In other words, an event causes an action to execute that changes the value(s) of an object's attribute(s) in a significant manner. The

¹¹ The approach described in this section are adapted from Booch, Rumbaugh, and Jacobson, *The Unified Modeling Language User Guide*.

A State:	
An Initial State:	
A Final State:	
An Event:	 anEvent
A Transition:	
A Frame:	

FIGURE 8-12 Behavioral State Machine Diagram Syntax**FIGURE 8-13**
Steps for Building
Behavioral State
Machine

1. Set the context.
2. Identify the initial, final, and stable states of the object.
3. Determine the order in which the object will pass through the stable states.
4. Identify the events, actions, and guard conditions associated with the transitions.
5. Validate the behavioral state machine.

actions are typically operations contained within the object. Also, guard conditions can model a set of test conditions that must be met for the transition to occur. At this point in the process, the transitions are drawn between the relevant states and labeled with the event, action, or guard condition.

The fifth step is to validate the behavioral state machine by making sure that each state is reachable and that it is possible to leave all states except for final states. Obviously, if an identified state is not reachable, either a transition is missing or the state was identified in error. Furthermore, only final states can be a dead end from the perspective of an object-life cycle.

Applying the Concepts at CD Selections

Basically, the only way to really understand the behavioral state machine-creation process is to attempt to draw one. As in the previous example diagrams, we focus our attention on the Place Order use case. In this case, we have picked the Order class to investigate.

The second step is to identify the various states that an order will have over its lifetime. To enable the discovery of the initial, final, and stable states of an order, the CD Selections team interviewed a customer representative that dealt with processing customer orders on a regular basis. Based on this interview, the team uncovered the life of an order (see Figure 8-14) from start to finish, from an order's perspective.

The third step is to determine the sequence of the states that an order object will pass through during its lifetime. Based on the order's lifecycle portrayed in Figure 8-14, the team identified and laid out the states of the order on the behavioral state machine.

Next, the team identified the events, actions, and guard conditions associated with the transitions between the states of the order. For example, the event "Order is created" moves the order from the "initial" state to the "In Process" state (see Figure 8-15). During the "Processing" state, a credit authorization is requested. The guard condition "Authorization = Approved" prevents the order to move from the "Processing" state to the "Placed" state unless the credit authorization has been approved. Also, the guard condition "Authorization = Denied" prevents the order to move from the "Processing" state to the "Denied" state unless the credit authorization has been denied. As such, between the two guard conditions, the order is stuck in the processing state until the credit authorization has been either approved or denied.

The team finally validates the behavioral state machine by ensuring that each state is reachable and that it is possible to leave all states except for the final states. Furthermore, the team makes sure that all states and transitions for the order have been modeled. At this point in time, one of the analysts on the team suggested that possibly there were multiple types of orders being described in the behavioral state machine. Specifically, he thought that there were denied and accepted orders. Based on this discovery, he suggested that two new classes, for each subtype of order, be created. However, upon further review by the entire team, it was decided that adding these classes to the class diagram and modifying all of the other diagrams to reflect this change would not add anything to the understanding of the problem. Therefore, it was decided not to add the classes. However, in many cases, modeling the states that an object will go through during its lifetime may in fact uncover additional useful subclasses. Figure 8-15 illustrates the behavioral state machine that the CD Selections team created for an order object.

1. The customer creates an order on the Web
2. The customer submits the order once he or she is finished
3. The credit authorization needs to be approved for the order to be accepted
4. If denied, the order is returned to the customer for changes or deletion
5. If accepted, the order is placed
6. The order is shipped to the customer
7. The customer receives the order
8. The order is closed

FIGURE 8-14
The Life of an Order

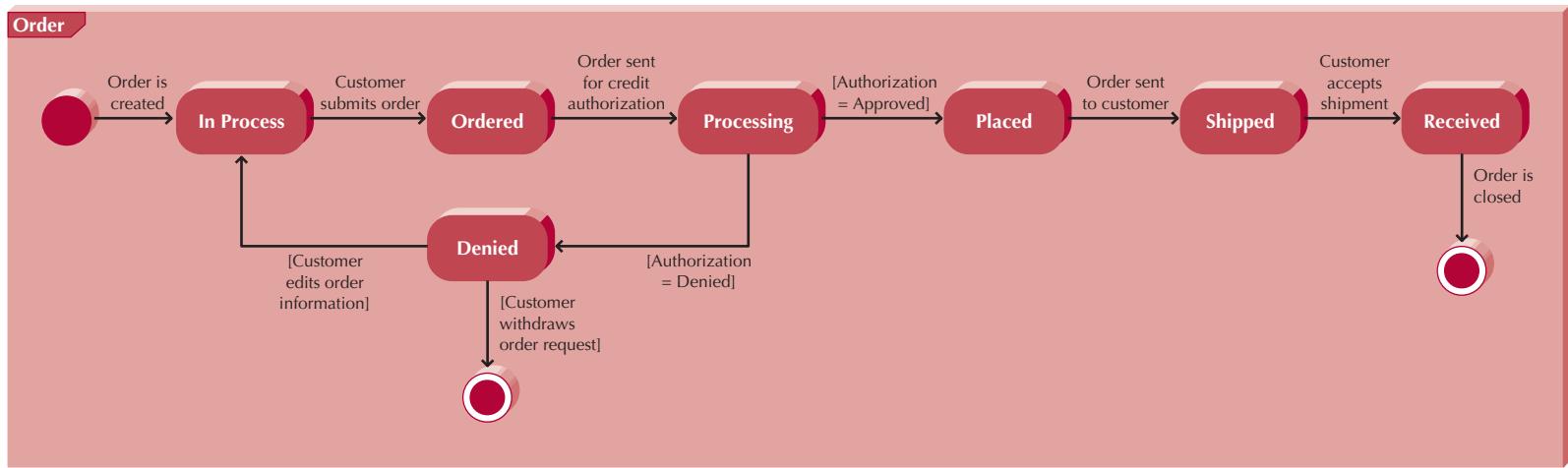


FIGURE 8-15 Behavioral State Machine for the Order Class

YOUR TURN

8-3 Drawing a Behavioral State Machine

You have been working with the system for the campus housing service that helps students find apartments. One of the dynamic classes in this system likely is the apartment class. Draw a behavioral state machine to

show the various states that an apartment class transitions throughout its lifetime. Can you think of other classes that would make good candidates for a behavioral state machine?

SUMMARY

Sequence Diagram

The sequence diagram is a dynamic model that illustrates the classes that participate in a use case and messages that pass between them over time. Classes are placed horizontally across the top of a sequence diagram, each having a dotted line called a lifeline vertically below it. The execution occurrence, represented by a thin rectangle, is placed over the lifeline to show when the classes are sending or receiving messages. A message is a communication between classes that conveys information with the expectation that activity will ensue, and the messages are shown using an arrow connecting two objects that points in the direction that the message is being passed.

To create a sequence diagram, first identify the context of the diagram: system, use case, scenario, or an operation. Next, identify the classes that participate in the context and then add the messages that pass among them. Finally, you will need to add the lifeline and focus of control. Sequence diagrams are helpful for understanding the time order of the messages.

Communication Diagram

Like the sequence diagram, the communication diagram provides a dynamic view of an object-oriented system. A communication diagram is similar to an object diagram except that it shows message-passing relationships instead of aggregation or generalization associations. The layout of the diagram is done in a manner that accentuates the message passing between the collaborating actors and objects. Undirected lines portray associations that are used to represent the collaborations between the actors and object. Messages are shown as labels on the associations.

When building a communication diagram, like a sequence diagram, you should first identify the context of the diagram: system, use case, scenario, or an operation. Next, identify the classes that participate in the context and then identify the associations that represent the collaborations between them. Finally, add the messages to the associations. Communication diagrams are helpful for understanding how objects collaborate.

Behavioral State Machine

The behavioral state machine shows the different states that a single class passes through during its life in response to events, along with responses and actions. A state is a set of values that describes an object at a specific point in time, and it represents a point in an object's life in which it satisfies some condition, performs some action, or waits for something to happen. An event is something that takes place at a certain point in time and

changes a value(s) that describes an object, which, in turn, changes the object's state. As objects move from state to state, they undergo transitions.

When drawing a behavioral state machine, like the other behavioral diagrams, the first thing is to set the context of the diagram: system, subsystem, set of classes, or an individual class. Then, rectangles with rounded corners are placed on the model to represent the various states on which the context will take. Next, arrows are drawn between the rectangles to denote the transitions, and event labels are written above the arrows to describe the event that causes the transition to occur. Typically, behavioral state machines are used to portray the dynamic aspect of a complex class.

KEY TERMS

Action	CRUD analysis	Object
Activity	CRUD matrix	Operation
Actor	Dynamic model	Operation call message
Association	Event	Packages
Attributes	Execution occurrence	Return message
Behavior	Final state	Scenario
Behavioral models	Frame	Self-delegation
Behavior state machines	Generic sequence diagram	Sequence diagram
Class	Guard condition	State
Class diagram	Initial state	State symbol
Collaboration	Instance	Temporary object
Communication diagram	Instance sequence diagram	Transition
Condition	Lifeline	Trigger
CRC cards	Message	Use case

QUESTIONS

- How is behavioral modeling related to structural modeling?
- How does a use case relate to a sequence diagram? A communication diagram?
- Contrast the following sets of terms:
state; behavior
class; object
action; activity
use case; scenario
method; message
- Why is iteration important when creating a behavioral model?
- Describe the main building blocks for the sequence diagram and how they are represented on the model.
- How do you show that a temporary object is to go out of existence on a sequence diagram?
- Do lifelines always continue down the entire page of a sequence diagram? Explain.
- Describe the steps used to create a sequence diagram.
- Describe the main building blocks for the communication diagram and how they are represented on the model.
- How do you show the sequence of messages on a communication diagram?
- How do you show the direction of a message on a communication diagram?
- Describe the steps used to create a communication diagram.
- Are states always depicted using rounded rectangles on a behavioral state machine? Explain.
- What kinds of events can lead to state transitions on a behavioral state machine?
- What are the steps in building a behavioral state machine?
- How are guard conditions shown on a behavioral state machine?
- Describe the type of class that is best represented by a behavioral state machine. Give two examples of classes that would be good candidates for a behavioral state machine.

- 18.** Identify the model(s) that contains each of the following components:

actor
association
class
extends association
final state
guard condition

initial state
links
message
multiplicity
object
state
transition
update operation

EXERCISES

- A.** Create a sequence diagram for each of the following scenario descriptions for a video store system. A Video Store (AVS) runs a series of fairly standard video stores.

1. Every customer must have a valid AVS customer card in order to rent a video. Customers rent videos for three days at a time. Every time a customer rents a video, the system must ensure that the customer does not have any overdue videos. If so, the overdue videos must be returned and an overdue fee paid before customer can rent more videos.
2. If the customer has returned overdue videos, but has not paid the overdue fee, the fee must be paid before new videos can be rented. If the customer is a premier customer, the first two overdue fees can be waived, and the customer can rent the video.
3. Every morning, the store manager prints a report that lists overdue videos; if a video is two or more days overdue, the manager calls to remind the customer to return the video.

- B.** Create a communication diagram for the video system in Exercise A.

- C.** Create a sequence diagram for each of the following scenario descriptions for a health club membership system.

1. When members join the health club, they pay a fee for a certain length of time. The club wants to mail out reminder letters to members asking them to renew their memberships one month before their memberships expire. About half of the members do not renew their memberships. These members are sent follow-up surveys to complete about why they decided not to renew so that the club can learn how to increase retention. If the member did not renew because of cost, a special discount is offered to that customer. Typically, 25 percent of accounts are reactivated because of this offer.
2. Every time a member enters the club, an attendant takes his or her card and scans it to make sure the

person is an active member. If the member is not active, the system presents the amount of money it costs to renew the membership. The customer is given the chance to pay the fee and use the club, and the system makes note of the reactivation of the account so that special attention can be given to this customer when the next round of renewal notices are dispensed.

- D.** Create a communication diagram for each of the health club membership scenarios described in Exercise C.
- E.** Think about sending a first-class letter to an international pen pal. Describe the process that the letter goes through to get from your initial creation of the letter to being read by your friend, from the letter's perspective. Draw a behavioral state machine that depicts the states that the letter moves through.
- F.** Consider the video store that is described in Question A. Draw a behavioral state machine that describes the various states that a video goes through from the time it is placed on the shelf through the rental and return process.
- G.** Draw a behavioral state machine that describes the various states that a travel authorization can have through its approval process. A travel authorization form is used in most companies to approve travel expenses for employees. Typically, an employee fills out a blank form and sends it to his or her boss for a signature. If the amount is fairly small ($\leq \$300$), then the boss signs the form and routes it to accounts payable to be input into the accounting system. The system cuts a check that is sent to the employee for the right amount, and after the check is cashed, the form is filed away with the cancelled check. If the check is not cashed within ninety days, the travel form expires. When the amount of the travel voucher is a large amount ($> \$300$), then the boss signs the form and sends it to the CFO along with a paragraph explaining the purpose of the travel, and the CFO will sign the form and pass it along to accounts

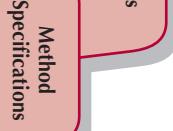
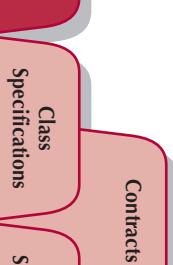
- payable. Of course, both the boss and the CFO can reject the travel authorization form if they do not feel that the expenses are reasonable. In this case, the employee can change the form to include more explanation, or decide to pay the expenses.
- H.** Picnics R Us (PRU) was described in Chapter 6. You have already created an activity diagram (Exercise P of Chapter 6), the use cases (Exercise P of Chapter 6), a use case diagram (Exercise Q of Chapter 6), and a structural model (Exercise P of Chapter 7). For this exercise,
1. Choose one use case and create a sequence diagram.
 2. Create a communication diagram for the use case chosen in question 1.
 3. Create a behavioral state machine to depict one of the classes on the class diagram you created for Exercise P of Chapter 7.
- I.** Of-the-Month Club (OTMC) was described in Chapter 6. You have already created an activity diagram (Exercise R of Chapter 6), the use cases (Exercise R of Chapter 6), a use case diagram (Exercise S of Chapter 6), and a structural model (Exercise Q of Chapter 7). For this exercise:
1. Choose one use case and create a sequence diagram.
 2. Create a communication diagram for the use case chosen in question 1.
 3. Create a behavioral state machine to depict one of the classes on the class diagram you created for Exercise Q of Chapter 7.
- J.** Think about your school or local library and the processes involved in checking out books, signing up new borrowers, and sending out overdue notices from the library's perspective. Describe three use cases that represent these three functions.
1. Create the use case diagram for the library system.
 2. Create a class diagram for the library system.
3. Choose one use case and create a sequence diagram.
4. Create a communication diagram for the use case chosen in question 3.
5. Create a behavioral state machine to depict one of the classes on the class diagram in question 2.
- K.** Think about the system that handles student admissions at your university. The primary function of the system should be able to track a student from the request for information through the admissions process until the student is either admitted or rejected from attending the school. An admissions form includes the contents of the form, SAT information, and references. Additional information is captured about students of alumni, such as their parent's graduation year, contact information, and college major. Pretend that a temporary student object is used by the system to hold information about people before they send in an admission form. After the form is sent in, these people are considered "students."
1. Create an activity diagram that describes this process.
 2. Based on the activity diagram, create a use case diagram for the process.
 3. Choose one of the use cases on the use case diagram and create an essential, detail use case description of it.
 4. Create a sequence diagram for the use case chosen in question 3.
 5. Create a communication diagram for the use case chosen in question 3.
 6. Create a class diagram for the admission system.
 7. Create a behavioral state machine to depict a person as he or she moves through the admissions process.

MINICASES

1. Refer to the use case descriptions and use-case diagram you prepared for Professional and Scientific Staff Management (PSSM) for Minicase 2 in Chapter 6. PSSM was so satisfied with your work that they wanted the behavioral models created so that they could understand the interaction that would take place between the users and the system in greater detail.
 - a. Create both a sequence and a communication diagram for each use case.
 - b. Based on the sequence and communication diagrams, create CRC cards for each class identified and a class diagram.
2. Refer to the structural model that you created for Holiday Travel Vehicles for Minicase 2 in Chapter 7.
 - a. Choose one of the more complex classes and create a behavioral state machine for it.
 - b. Based on the structural model you created, role play the CRC cards to develop a set of use cases that describe a typical sales process.
 - c. Create both a sequence and communication diagram for the use case.

CHAPTER 9 ■

MOVING ON
TO DESIGN



PART THREE

DESIGN PHASE

The Design Phase decides *how* the system will operate. First, the project team creates a Design Plan and a set of factored and partitioned Analysis models (Functional, Structural, and Behavioral). The class and method designs are illustrated using the Class Specifications (using CRC Cards and Class Diagrams), Contracts, and Method Specifications. Next, the data management layer is addressed by designing the actual database or file structure to be used for object persistence and a set of classes that will map the Class Specifications into the object persistence format chosen. Next, the team produces the user interface layer design using Use Scenarios, Windows Navigation Diagrams, Real Use Cases, Interface Standards, and User Interface Templates. The physical architecture layer design is created using a Deployment Diagrams and Hardware/Software Specification. This collection of deliverables is the system specification that is handed to the programming team for implementation. At the end of the Design Phase, the feasibility analysis and project plan are reexamined and revised, and another decision is made by the project sponsor and approval committee about whether to terminate the project or continue.

CHAPTER 10 ■

CLASS AND
METHOD DESIGN

CHAPTER 11 ■

DATA
MANAGEMENT
LAYER DESIGN

CHAPTER 12 ■

HUMAN COMPUTER
INTERACTION
LAYER DESIGN

CHAPTER 13 ■

PHYSICAL
ARCHITECTURE
LAYER DESIGN

CHAPTER 9

MOVING ON TO DESIGN

The design phase in object-oriented system development uses the requirements that were gathered during analysis to create a blueprint for the future system. A successful design builds upon what was learned in earlier phases and leads to a smooth implementation by creating a clear, accurate plan of what needs to be done. This chapter describes the initial transition from analysis to design and presents three ways to approach the design for the new system.

OBJECTIVES

- Understand the transition from analysis to design.
- Understand the use of factoring, partitions, and layers.
- Be able to create package diagrams.
- Be familiar with the custom, packaged, and outsource design alternatives.
- Be able to create an alternative matrix.

CHAPTER OUTLINE

Introduction	Design Strategies
Evolving the Analysis Models into Design Models	Custom Development
Factoring	Packaged Software
Partitions and Collaborations	Outsourcing
Layers	Selecting a Design Strategy
Packages and Package Diagrams	Developing the Actual Design
Identifying Packages and Creating Package Diagrams	Alternative Matrix
Applying the Concepts at CD Selections	Applying the Concepts at CD Selections
	Summary

INTRODUCTION

The purpose of the analysis phase is to figure out *what* the business needs. The purpose of the *design phase* is to decide *how* to build it. The major activity that takes place during the design phase is evolving the set of analysis representations into design representations.

Throughout the design phase, the project team carefully considers the new system in respect to the current environment and systems that exist within the organization as a whole. Major considerations of the “*how*” of a system are environmental factors like integrating with existing systems, converting data from legacy systems, and leveraging skills that exist in-house. Although the Planning and Analysis phases are undertaken to develop

a “possible” system, the goal of the Design phase is to create a blueprint for a system that makes sense to implement.

An important initial part of the design phase is to examine several design strategies and decide which will be used to build the system. Systems can be built from scratch, purchased and customized, or outsourced to others, and the project team needs to investigate the viability of each alternative. The decision to make, to buy, or to outsource influences the design tasks that are accomplished throughout the rest of the phase.

At the same time, detailed design of the individual classes and methods that are used to map out the nuts and bolts of the system and how they are to be stored must still be completed. Techniques like CRC cards, class diagrams, contract specification, method specification, and database design provide detail in preparation for the implementation phase, and they ensure that programmers have sufficient information to build the right system efficiently. These topics are covered in Chapters 10 and 11.

Design also includes activities like designing the user interface, system inputs, and system outputs, which involve the ways that the user interacts with the system. Chapter 12 describes these three activities in detail, along with techniques, such as story boarding and prototyping that help the project team design a system that meets the needs of its users and is satisfying to use.

Finally, physical architecture decisions are made regarding the hardware and software that will be purchased to support the new system and the way that the processing of the system will be organized. For example, the system can be organized so that its processing is centralized at one location, distributed, or both centralized and distributed, and each solution offers unique benefits and challenges to the project team. Global issues and security need to be considered along with the system’s technical architecture because they will influence the implementation plans that are made. Physical architecture, security, and global issues will be described in Chapter 13.

The many steps of the design phase are highly interrelated and, as with the steps in the analysis phase, the analysts often go back and forth among them. For example, prototyping in the interface design step often uncovers additional information that is needed in the system. Alternatively, a system that is being designed for an organization that has centralized systems may require substantial hardware and software investments if the project team decides to change to a system in which all of the processing is distributed.

In this chapter, we overview the processes that are used to evolve the analysis models into design models. Next, we introduce the use of packages and package diagrams. Finally, we examine the three fundamental approaches to developing new systems: make, buy, or outsource.

EVOLVING THE ANALYSIS MODELS INTO DESIGN MODELS

The purpose of the analysis models was to represent the underlying business problem domain as a set of collaborating objects. In other words, the analysis activities defined the functional requirements. To achieve this, the analysis activities ignored nonfunctional requirements such as performance and the system environment issues (e.g., distributed or centralized processing, user interface issues, and database issues). In contrast, the primary purpose of the design models is to increase the likelihood of successfully delivering a system that implements the functional requirements in a manner that is affordable and easily maintainable.

Therefore, in system design, we address both the functional and nonfunctional requirements. From an object-oriented perspective, system design models simply refine the system analysis models by adding system environment (or solution domain) details to them and refining the problem domain information already contained in the analysis models.

**PRACTICAL****TIP****Avoiding Classic Design Mistakes**

In Chapters 3 and 4, we discussed several classic mistakes and how to avoid them. Here, we summarize four classic mistakes in the design phase, and discuss how to avoid them.

1. Reducing design time: If time is short, there is a temptation to reduce the time spent in “unproductive” activities such as design so that the team can jump into “productive” programming. This results in missing important details that have to be investigated later at a much higher time cost (usually at least ten times longer).

Solution: If time pressure is intense, use timeboxing to eliminate functionality or move it into future versions.

2. Feature creep: Even if you are successful at avoiding scope creep, about 25 percent of system requirements will still change. And, changes—big and small—can significantly increase time and cost.

Solution: Ensure that all changes are vital and that the users are aware of the impact on cost and time. Try to move proposed changes into future versions.

3. Silver bullet syndrome: Analysts sometimes believe the marketing claims for some design tools that claim to solve all problems and magically reduce time and costs. No one tool or technique can eliminate overall time or costs by more than 25 percent (although some can reduce individual steps by this much).

Solution: If a design tool has claims that appear too good to be true, just say no.

4. Switching tools in mid-project: Sometimes analysts switch to what appears to be a better tool during design in the hopes of saving time or costs. Usually, any benefits are outweighed by the need to learn the new tool. This also applies to even “minor” upgrades to current tools.

Solution: Don’t switch or upgrade unless there is a compelling need for specific features in the new tool, and then explicitly increase the schedule to include learning time.

Adapted from Steve McConnell, *Rapid Development*, Microsoft Press, Redmond, WA 1996.

When evolving the analysis model into the design model, you should first carefully review the use cases and the current set of classes (their methods and attributes, and the relationships between them). Are all of the classes necessary? Are there any missing classes? Are the classes fully defined? Are there any missing attributes or methods? Do the classes have any unnecessary attributes and methods? Is the current representation of the evolving system optimal?

In the following sections, we introduce factoring, partitions and collaborations, and layers as a way to evolve problem domain-oriented analysis models into optimal solution domain-oriented design models.

Factoring

Factoring is the process of separating out a *module* into a standalone module in and of itself. The new module can be a new *class* or a new *method*. For example, when reviewing a set of classes, it may be discovered that they have a similar set of attributes and methods. As such, it may make sense to factor out the similarities into a separate class. Depending on whether the new class should be in a superclass relationship to the existing classes or not, the new class can be related to the existing classes through *generalization (A-Kind-Of)* or possibly through *aggregation (Has-Parts)* relationship. For example, using the appointment system example in the previous chapters (see Figure 7-2), if the Employee class had not been identified, we could possibly identify it at this stage by factoring out the similar methods and attributes from the Nurse, Administrative Staff, and Doctor classes.

In this case, we would relate the new class (Employee) to the existing classes using the generalization (A-Kind-Of) relationship.

Abstraction and *refinement* are two closely related processes to factoring. Abstraction deals with the creation of a “higher” level idea from a set of ideas. Identifying the Employee class is an example of abstracting from a set of lower classes to a higher one. In some cases, the abstraction process will identify *abstract classes*, whereas, in other situations, it will identify additional *concrete classes*.¹ The refinement process is the opposite of the abstraction process. In the appointment system example in the previous chapters (see Figure 7-2), we could identify additional subclasses of the Administrative Staff class, such as Receptionist, Secretary, and Bookkeeper. Of course we would only add the new classes if there were sufficient differences between them. Otherwise, the more general class, Administrative Staff, would suffice.

Partitions and Collaborations

Based on all of the factoring, refining, and abstracting that can take place to the evolving system, the sheer size of the system representation can overload both the user and the developer. At this point in the evolution of the system, it may make sense to split the representation into a set of *partitions*. A partition is the object-oriented equivalent of a subsystem,² where a subsystem is a decomposition of a larger system into its component systems (e.g., an accounting information system could be functionally decomposed into an accounts payable system, an account receivable system, a payroll system, and so on). From an object-oriented perspective, partitions are based on the pattern of activity (messages sent) among the objects in an object-oriented system.

A good place to look for potential partitions is the *collaborations* modeled in UML’s communication diagrams (see Chapter 8). If you recall, one useful way to identify collaborations is to create a communication diagram for each use case. However, since an individual class may support multiple use cases, an individual class can participate in multiple use case-based collaborations. In those cases where classes are supporting multiple use cases, the collaborations should be merged together. Also, CRUD analysis (see Chapter 8) can be used to identify potential classes on which to merge collaborations.

Depending on the complexity of the merged collaboration, it may be useful in decomposing the collaboration into multiple partitions. In this case, in addition to having collaborations between objects, it is possible to have collaborations among partitions. The general rule of thumb is the more messages sent between objects, the more likely the objects belong in the same partition. The fewer messages sent, the less likely the two objects belong together.

Another useful approach to identify potential partitions is to model each collaboration between objects in terms of clients, servers, and contracts. A *client* is an instance of a class that sends a *message* to an instance of another class for a *method* to be executed; a *server* is the instance of a class that receives the message; and a *contract* is the specification that formalizes the interactions between the client and server objects (see Chapters 7 and 10). This approach allows the developer to build up potential partitions by looking at the contracts that have been specified between objects. In this case, the more contracts there are between objects, the more often than not the objects belong in the same partition. The fewer contracts, the chances are the two classes do not belong in the same partition.

¹ See Chapter 7 for the differences between abstract and concrete classes.

² Some authors refer to partitions as subsystems (e.g., see Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Weiner, *Designing Object-Oriented Software* [Englewood Cliffs, NJ: Prentice-Hall, 1990]), while others refer to them as layers (e.g., see Ian Graham, *Migrating to Object Technology* [Reading, MA Addison-Wesley, 1994]). However, we have chosen to use the term partition (from Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design* [Englewood Cliffs, NJ: Prentice Hall, 1998]) to minimize confusion between subsystems in a traditional system development approach and the layers associated with Rational’s Unified Approach.

**YOUR
TURN**

9-1 Campus Housing

You have been working with the system for the campus housing service over the previous three chapters. In Chapter 6, you were asked to create a set of use cases (Your Turn 6-6) and to create a use case diagram (Your Turn 6-8). In Chapter 7 (Your Turn 7-3), you created a structural model (CRC cards and class diagram) for the same situation. In Chapter 8, you created a sequence diagram (Your Turn 8-1) and a communication diagram (Your Turn 8-2) for one of

the use cases you had identified in Chapter 6. Finally, you also created a behavioral state machine for the apartment class in Chapter 8 (Your Turn 8-3).

Based on the current set of functional, structural, and behavioral models portrayed in these diagrams, apply the abstraction and refinement processes to identify additional abstract and concrete classes that would be useful to include in the evolving system.

Layers

Until this point in the development of our system, we have focused only on the problem domain; we have totally ignored the system environment (physical architecture, user interface, and data access and management). To successfully evolve the analysis model of the system into a design model of the system, we must add the system environment information. One useful way to do this, without overloading the developer, is to use *layers*. A layer represents an element of the software architecture of the evolving system. We have focused only on one layer in the evolving software architecture: the problem domain layer. There should be a layer for each of the different elements of the system environment (e.g., system architecture, user interface, and data access and management).

The idea of separating the different elements of the architecture into separate layers can be traced back to the MVC architecture of *Smalltalk*.³ When Smalltalk was first created,⁴ the authors decided to separate the application logic from the logic of the user interface. In this manner, it was possible to easily develop different user interfaces that worked with the same application. To accomplish this, they created the *Model-View-Controller* (MVC) architecture where *Models* implemented the application logic (problem domain), and *Views* and *Controllers* implemented the logic for the user interface. Views handled the output and Controllers handled the input. Since graphical user interfaces were first developed in the Smalltalk language, the MVC architecture served as the foundation for virtually all graphical user interfaces that have been developed today (including the Mac interfaces, the Windows family, and the various Unix-based GUI environments).

Based on Smalltalk's innovative MVC architecture, many different software layers have been proposed.⁵ Based on these proposals, we suggest the following layers on which to base

³ See Simon Lewis, *The Art and Science of Smalltalk: An Introduction to Object-Oriented Programming Using VisualWorks* (Englewood Cliffs, NJ: Prentice-Hall, 1995).

⁴ Smalltalk was first invented in the early 1970s by a software development research team at Xerox PARC. It introduced many new ideas into the area of programming languages (e.g., object-orientation, windows-based user interfaces, reusable class library, and the idea of a development environment). In many ways, Smalltalk is the father (or mother) of all object-based and object-oriented languages, such as Visual Basic, C++, and Java.

⁵ For example, Problem Domain, Human Interaction, Task Management, and Data Management (Peter Coad and Edward Yourdon, *Object-Oriented Design* [Englewood Cliffs, NJ: Yourdon Press, 1991]); Domain, Application, and Interface (Ian Graham, *Migrating to Object Technology* [Reading, MA: Addison Wesley, 1994]); Domain, Service, and Presentation (Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design* [Englewood Cliffs, NJ: Prentice Hall, 1998]); Business, View, and Access (Ali Bahrami, *Object-Oriented Systems Development using the Unified Modeling Language*, [New York, NY: McGraw Hill, 1999]); Application-Specific, Application-General, Middleware, System-Software (Ivar Jacobson, Grady Booch, and James Rumbaugh, *The Unified Software Development Process* [Reading, MA: Addison-Wesley, 1999]); and Foundation, Architecture, Business, and Application (Meilir Page-Jones, *Fundamentals of Object-Oriented Design in UML* [Reading, MA: Addison-Wesley, 2000]).

software architecture: foundation, physical architecture, human computer interaction, data access and management, and problem domain (see Figure 9-1). Each layer limits the types of classes that can exist on it (e.g., only user interface classes may exist on the human computer interaction layer). The following provides a short description of each layer.

Foundation The *foundation layer* is, in many ways, a very uninteresting layer. It contains classes that are necessary for any object-oriented application to exist. They include classes that represent fundamental data types (e.g., integers, real numbers, characters, and strings), classes that represent fundamental data structures (sometimes referred to as container classes; e.g., lists, trees, graphs, sets, stacks, and queues), and classes that represent useful abstractions (sometimes referred to as utility classes; e.g., date, time, and money). Today, the classes found on this layer typically are included with the object-oriented development environments.

Physical Architecture The *physical architecture layer* addresses how the software will execute on specific computers and networks. As such, this layer includes classes that deal with communication between the software and the computer's operating system and the network. For example, classes that address how to interact with the various ports on a specific computer would be included in this layer. This layer also includes classes that would interact with so-called *middleware* applications, such as the OMG's CORBA and Microsoft's DCOM architectures that handle distributed objects.

Unlike the foundation layer, there are many design issues that must be addressed before choosing the appropriate set of classes for this layer. These design issues include the choice of a computing or network architecture (such as the various client-server architectures), the actual design of a network, hardware and server software specification, global/international issues (such as multilingual requirements), and security issues. A complete description of all of the issues related to system architecture is beyond the scope of this book—there are entire courses dedicated to this subject. However, we do present the basic issues in Chapter 13.

Human Computer Interaction The *human computer interaction layer* contains classes associated with the View and Controller idea from Smalltalk. The primary purpose of this layer is to keep the specific user interface implementation separate from the problem domain classes. This increases the portability of the evolving system. Typical classes found on this layer include classes that can be used to represent buttons, windows, text fields, scroll bars, check boxes, drop-down lists, and many other classes that represent user interface elements.

When it comes to designing the user interface for an application, there are many issues that must be addressed. For example, how important is consistency across different user interfaces, what about differing levels of user experience, how is the user expected to be able to navigate through the system, what about help systems and online manuals, what types

Layers	Examples	Relevant Chapters
Foundation	Date, Enumeration	9, 10
Physical Architecture	ServerSocket, URLConnection	10, 13
Human Computer Interaction	Button, Panel	10, 12
Data Management	DataInputStream, FileInputStream	10, 11
Problem Domain	Employee, Customer	6, 7, 8, 9, 10

FIGURE 9-1
Layers and Sample
Classes

of input elements should be included (e.g., text box, radio buttons, check boxes, sliders, drop-down list boxes, etc.), and what types of output elements should be included (e.g., text, tables, graphs, etc.). Like the physical architecture layer, a complete description of all of the issues related to human computer interaction is beyond the scope of this book.⁶ However from the user's perspective, the user interface is the system. As such, we present the basic issues in user interface design in Chapter 12.

Data Management The *data management layer* addresses the issues involving the persistence of the objects contained in the system. The types of classes that appear in this layer deal with how objects can be stored and retrieved. Like the human computer interface layer, the classes contained in this layer allow the problem domain classes to be independent of the storage utilized, and hence increase the portability of the evolving system. Some of the issues related to this layer include choice of the storage format (such as relational, object/relational, and object databases) and storage optimization (such as clustering and indexing). A complete description of all of the issues related to the data management layer also is beyond the scope of this book.⁷ However, we do present the fundamentals in Chapter 11.

Problem Domain The previous layers dealt with classes that represent elements from the system environment and which will be added to the evolving system specification. The *problem domain layer* is what we have focused our attention on up until now. At this stage of the development of our system, we will need to further detail the classes so that it will be possible to implement them in an effective and efficient manner.

Many issues need to be addressed when designing classes, no matter on which layer they appear. For example, there are issues related to factoring, cohesion and coupling, concreteness, encapsulation, proper use of inheritance and polymorphism, constraints, contract specification, and detailed method design. These issues are discussed in Chapter 10.

PACKAGES AND PACKAGE DIAGRAMS

In UML, collaborations, partitions, and layers can be represented by a higher-level construct: a package.⁸ A *package* is a general construct that can be applied to any of the elements in UML models. In Chapter 6, we introduced the idea of packages as a way to group use cases together to make the use case diagrams easier to read and to keep the models at a reasonable level of complexity. In Chapters 7 and 8, we did the same thing for class and communication diagrams, respectively. In this section we describe a package diagram: a diagram that is composed only of packages. A *package diagram* is effectively a class diagram that only shows packages.

The symbol for a package is similar to a tabbed folder (see Figure 9-2). Depending on where a package is used, packages can participate in different types of relationships. For example, in a class diagram, packages represent groupings of classes. Therefore, aggregation and association relationships are possible.

⁶ One of the best books on user interface design is Ben Schneiderman, *Designing the User Interface: Strategies for Effective Human Computer Interaction*, 3rd Ed (Reading, MA: Addison-Wesley, 1998).

⁷ There are many good database design books that are relevant to this layer, see for example, Fred R. McFadden, Jeffrey A. Hoffer, Mary B. Prescott, *Modern Database Management*, 4th ed. (Reading, MA: Addison-Wesley, 1998), Michael Blaha and William Premerlani, *Object-Oriented Modeling and Design for Database Applications* (Englewood Cliffs, NJ: Prentice Hall, 1998), and Robert J. Muller, *Database Design for Smarties: Using UML for Data Modeling* (San Francisco, CA: Morgan Kaufmann, 1999).

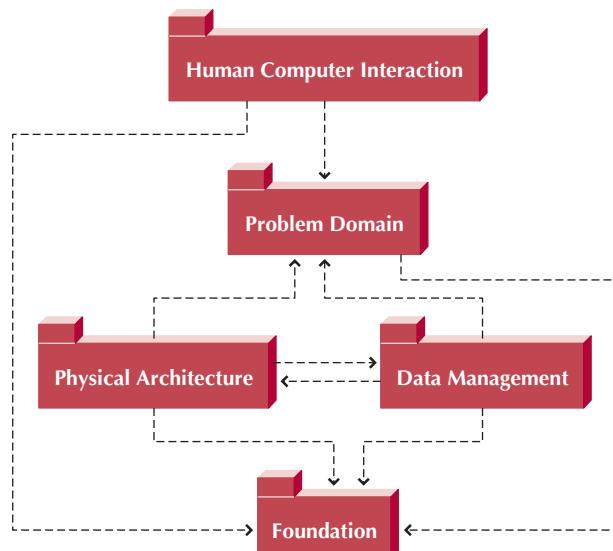
⁸ This discussion is based on material in Chapter 7 of Martin Fowler with Kendall Scott, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 2nd Ed. (Reading, MA: Addison Wesley, 2000).

<p>A Package:</p> <ul style="list-style-type: none"> ■ A logical grouping of UML elements ■ Used to simplify UML diagrams by grouping related elements into a single higher-level element 	 Package
<p>A Dependency Relationship:</p> <ul style="list-style-type: none"> ■ Represents a dependency between packages, i.e., if a package is changed, the dependent package also could have to be modified ■ The arrow is drawn from the dependent package toward the package on which it is dependent 	

FIGURE 9-2 Syntax for Package Diagram

In a package diagram, a new relationship, the *dependency relationship*, is useful to depict. A dependency relationship is portrayed by a dashed arrow (see Figure 9-2). A dependency relationship represents the fact that a modification dependency exists between two packages. That is, it is possible that a change in one package potentially could cause a change to be required in another package. Figure 9-3 portrays the dependencies among the different layers (foundation, physical architecture, human computer interaction, data access and management, and problem domain). For example, if a change occurs in the problem domain layer, it most likely will cause changes to occur in the human computer interaction, physical architecture, and data management layers. Notice that these layers “point to” the problem domain layer; as such, they are dependent on it. However, the reverse is not true.

At the class level, there could be many causes for dependencies among classes. For example, if the protocol for a method is changed, then this causes the interface for all objects of this class to change. Therefore, all classes that have objects that send messages to the instances of the modified class may have to be modified. Capturing dependency relationships among the classes and packages helps the organization in maintaining object-oriented information systems.

**FIGURE 9-3**
Package Diagram of
Dependency Relation-
ships among Layers

As already stated, collaborations, partitions, and layers are modeled as packages in UML. Furthermore, collaborations are normally factored into a set of partitions, which are typically placed on a layer. In addition, partitions can be composed of other partitions. Also, it is possible to have classes in partitions, which are contained in another partition, which is placed on a layer. All of these groupings are represented using packages in UML. Remember that a package is simply a generic, grouping construct used to simplify UML models through the use of composition.⁹

A simple package diagram, based on the appointment system example from the previous chapters, is shown in Figure 9-4. This diagram only portrays a very small portion of the entire system. In this case, we see that the Patient UI, DAM-Patient, and Patient Table classes are dependent on the Patient class. Furthermore, the DAM-Patient class is dependent on the Patient Table class. The same can be seen with the classes dealing with the actual appointments. By isolating the Problem Domain classes (such as the Patient and Appt classes) from the actual object persistence classes (such as the Patient Table and Appt Table classes) through the use of the intermediate Data Access and Manipulation classes (DAM-Patient and DAM-Appt classes), we isolate the Problem Domain classes from the actual storage medium.¹⁰ This greatly simplifies the maintenance and increases the reusability of the Problem Domain classes. Of course, in a complete description of a real system, there would be many more dependencies.

Identifying Packages and Creating Package Diagrams

In this section, we describe a simple five-step process to create package diagrams (see Figure 9-5). The first step is to set the context for the package diagram. Remember, packages can be used to model partitions and/or layers. Revisiting the appointment system again, let's set the context as the problem domain layer.

The second step is to cluster the classes together into partitions based on the relationships that the classes share. The relationships include generalization, aggregation, the various associations, and the message sending that takes place between the objects in the system. To identify the packages in the appointment system, we should look at the different analysis models (e.g., the class diagram [see Figure 7-2], sequence diagrams [see Figure 8-1], and the communication diagrams [see Figure 8-6]). Any classes in a generalization hierarchy should be kept together in a single partition.

The third step is to place the clustered classes together in a partition and model the partitions as packages. Figure 9-6 portrays five packages: PD Layer, Person Pkg, Patient Pkg, Appt Pkg, and Treatment Pkg.

The fourth step is to identify the dependency relationships among the packages. In this case, we review the relationships that cross the boundaries of the packages to uncover potential dependencies. In the appointment system, we see association relationships that connect the Person Pkg with the Appt Pkg (via the association between the Doctor class and the Appointment class), and the Patient Pkg, which is contained within the Person Pkg, with the Appt Pkg (via the association between the Patient and Appointment classes) and the Treatment Pkg (via the association between the Patient and Symptom classes).

The fifth step is to place the dependency relationships on the evolved package diagram. In the case of the Appointment system, there are dependency relationships between the Person Pkg and the Appt Pkg and the Person Pkg and the Treatment Pkg. To increase the

⁹ For those familiar with traditional approaches, such as structured analysis and design, packages serve a similar purpose as the leveling and balancing processes used in data flow diagramming.

¹⁰ These issues are described in more detail in Chapter 11.

understandability of the dependency relationships among the different packages, a pure package diagram that only shows the dependency relationships among the packages can be created (see Figure 9-7).

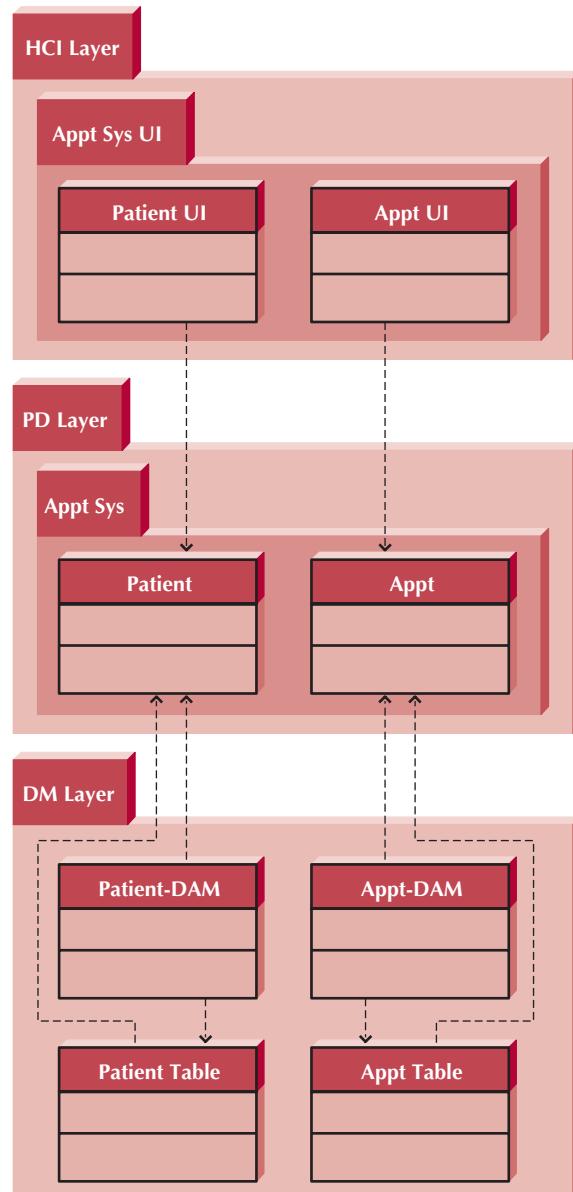


FIGURE 9-4
Partial Package Diagram of the Appointment System

FIGURE 9-5 Steps for Identifying Packages and Building Package Diagram

1. Set the Context.
2. Cluster classes together based on shared relationships.
3. Model clustered classes as a package.
4. Identify dependency relationships among packages.
5. Place dependency relationships between packages.

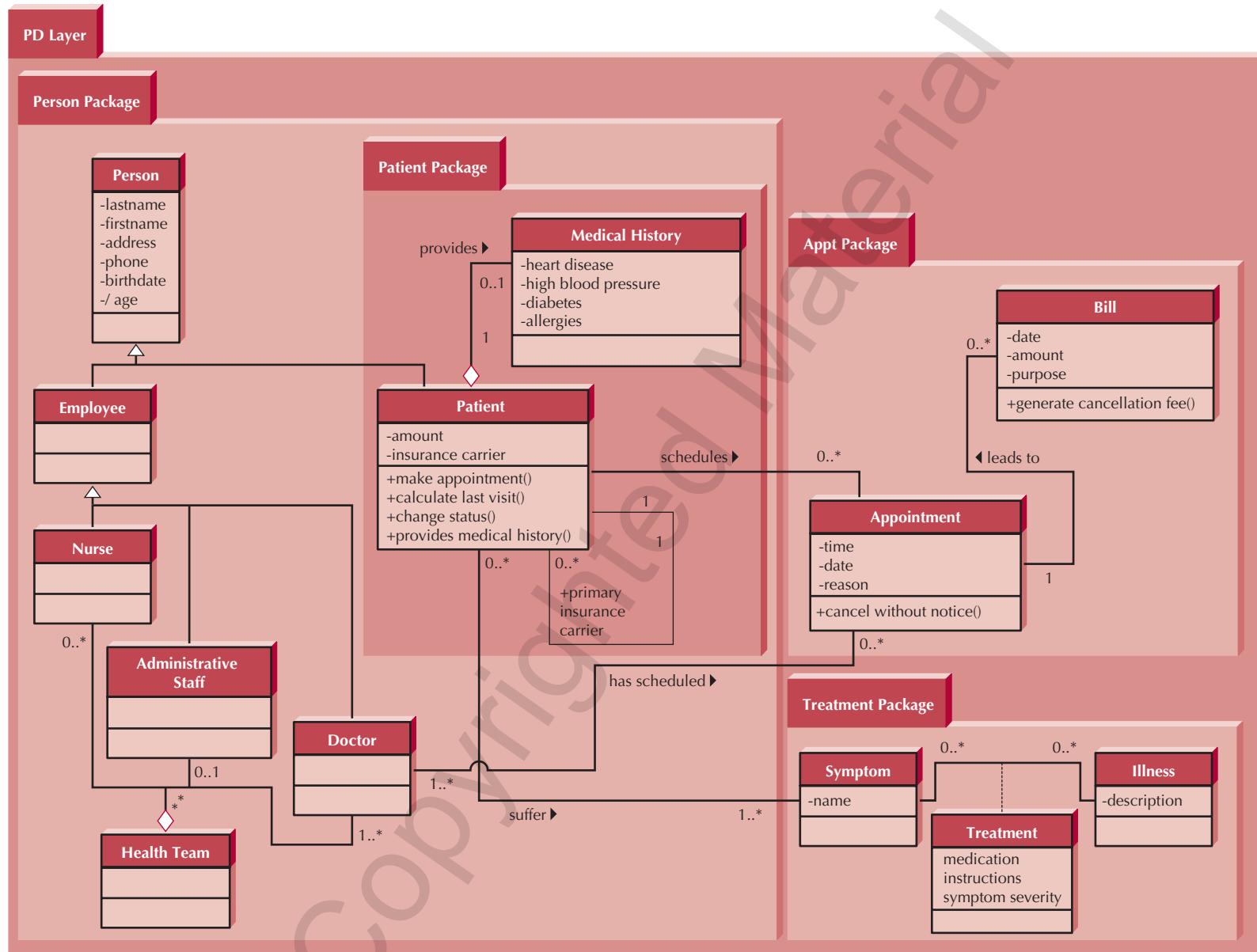


FIGURE 9-6 Package Diagram of the PD Layer for the Appointment System

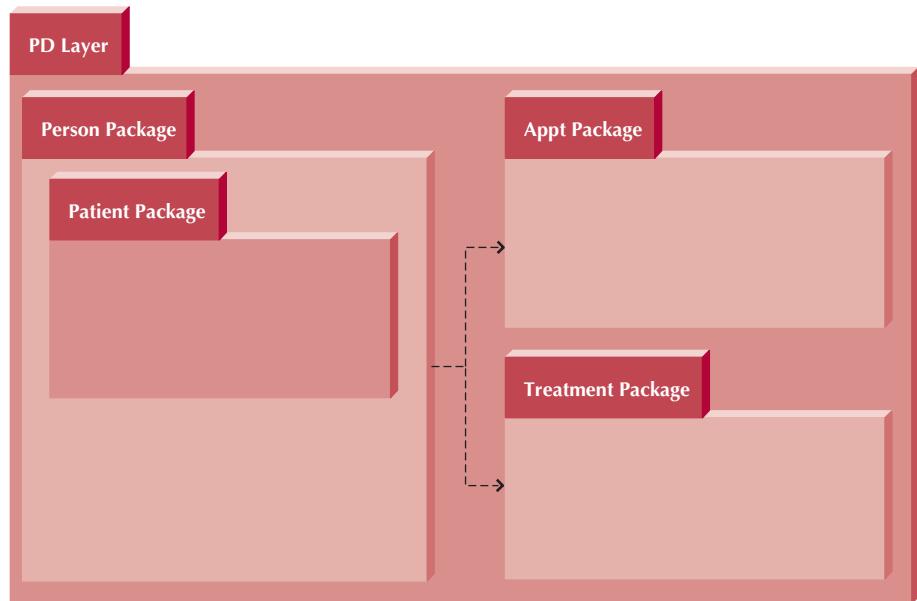


FIGURE 9-7
Overview Package Diagram of the PD Layer for the Appointment System

Applying Concepts at CD Selections

In the previous chapters, the CD Selections Internet sales system has been described. In Chapter 5, the functional and nonfunctional requirements for the system were given in Figure 5-13. In Chapter 6, the functional model in the form of an activity diagram (see Figure 6-12) and the use case descriptions and diagram (see Figures 6-13 through 6-17). The structural model comprised of the CRC cards and class diagram was given in Chapter 7 (see Figures 7-9 through 7-11 and 7-13). Finally, in Chapter 8 behavioral models were developed for the Places Order use case and the Order class (see the sequence diagram in Figure 8-5, the communication diagram in Figure 8-10, the behavioral state machine in Figure 8-15). In this section, we demonstrate the creation of a package diagram for the CD Selections Internet sales system.

As just detailed, the first thing to do when identifying packages and creating package diagrams is to set the context. At this point in the development of the Internet sales system for CD Selections, Alec wants the development team only to concentrate on the Problem Domain Layer.

The second step, cluster classes together, is accomplished by reviewing the relationships among the different classes (see Figures 7-13, 8-5, and 8-10). Through this review process, we see that there are generalization, aggregation, various associations, and message sending relationships. Since we always want to keep classes in a generalization hierarchy together, we automatically cluster the Customer, Individual, and Organization classes together to form a partition. It is also preferred to keep classes together that participate in aggregation relationships. Based on aggregation relationships, we cluster the Mkt Info, Review, Artist Info, and Sample Clip classes together in a partition. Based on the association relationship and the message-sending pattern of activity (contained in the communication diagram in Figure 8-10) between the CD and Mkt Info classes, it seemed to the development team that these classes should be in the same partition. Furthermore, since the Vendor class is only related to the CD class (see the class diagram in Figure 7-13), the development team decided to place it in the same partition. Finally, the development team decided to place the Order and Order Item classes together and the Search Req and CD List classes together in their own partitions.

The third step was to model each of these partitions as packages. Figure 9-8 shows the classes being contained in their respective packages. Observe that the Credit-Card Center currently is not contained in any package.

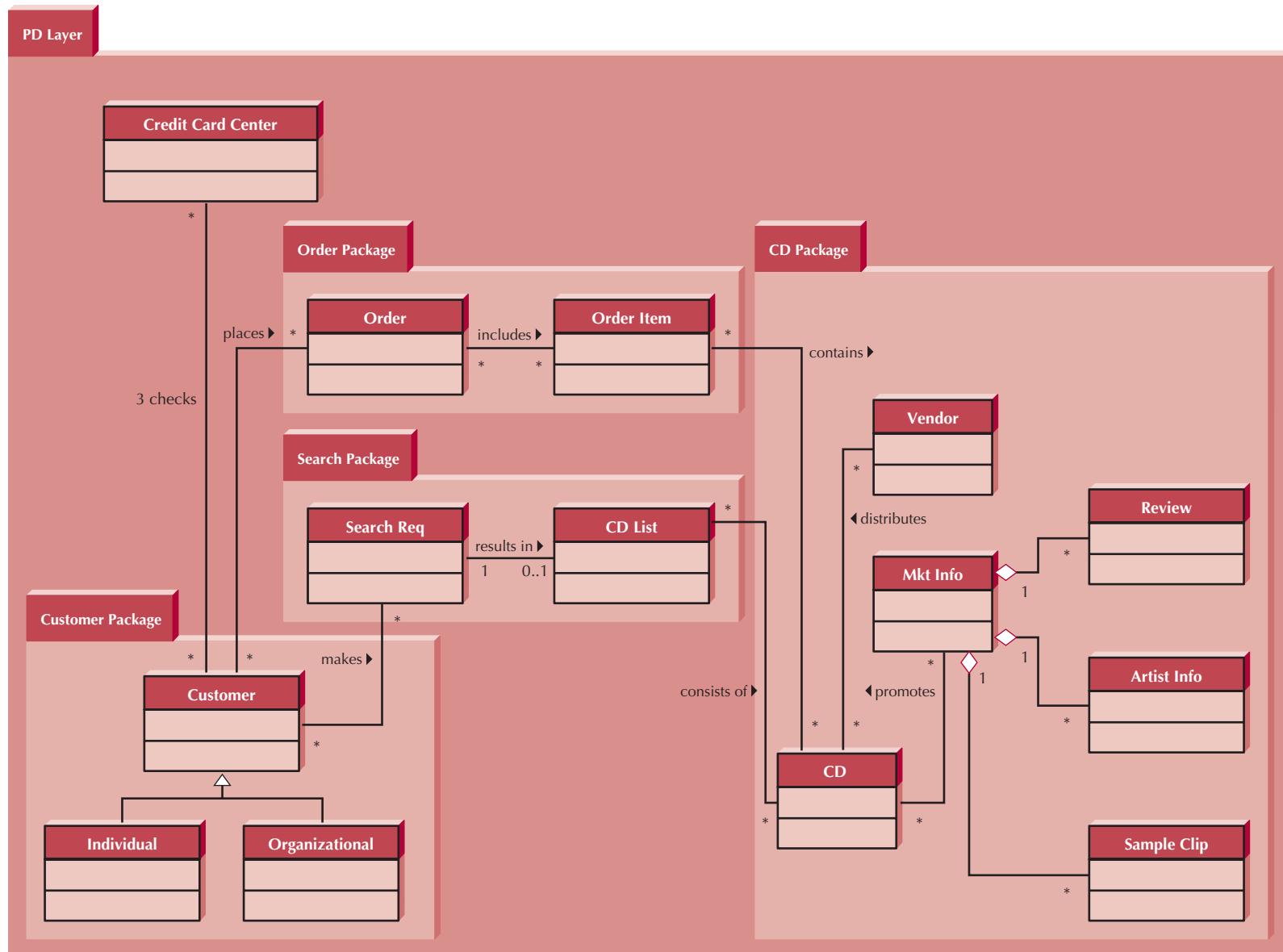


FIGURE 9-8 Package Diagram of the PD Layer of CD Selections Internet Sales System

Identifying dependency relationships among the packages is the fourth step. In this case, Alec was capable of quickly identifying four associations among the different packages: the Customer Package and the Order Package, the Customer Package and the Search Package, the Order Package and the CD Package, and the Search Package and the CD Package. He also identified an association between the Credit Card Clearance Center class and the Customer Package. Based on these associations, five dependency relationships were identified.

The fifth and final step is to place the dependency relationships on the package diagram. Again, to increase the understandability of the dependency relationships among the different packages, Alec decided to create a pure package diagram that only depicted the highest-level packages (and in this case the Credit Card Center class) and the dependency relationships (see Figure 9-9).

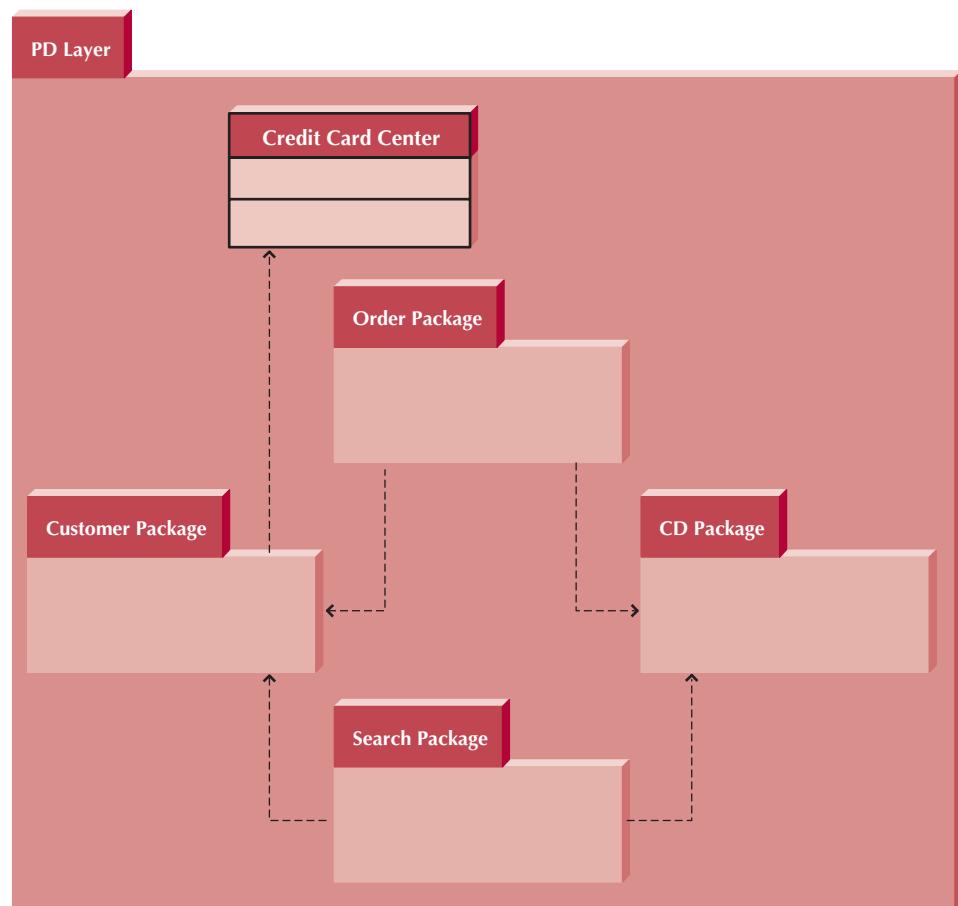


FIGURE 9-9
Overview Package
Diagram of the PD
Layer of CD Selections
Internet Sales System

**YOUR
TURN**

9-2 Campus Housing

Based on the factoring of the evolving system in Your Turn 9-1, identify a set of partitions for the Problem Domain layer and model them in a package diagram.

DESIGN STRATEGIES

Until now, we have assumed that the system will be built and implemented by the project team; however, there are actually three ways to approach the creation of a new system: developing a custom application in-house, buying a packaged system and customizing it, and relying on an external vendor, developer, or service provider to build the system. Each of these choices has its strengths and weaknesses, and each is more appropriate in different scenarios. The following sections describe each design choice in turn, and then we present criteria that you can use to select one of the three approaches for your project.

Custom Development

Many project teams assume that *custom development*, or building a new system from scratch, is the best way to create a system. For one thing, teams have complete control over the way the system looks and functions. Let's consider the order taking process for CD Selections. If the company wanted a Web order-taking feature that links tightly with its existing distribution system, the project may involve a complex, highly specialized program. Or, CD Selections might have a technical environment in which all information systems are built using standard technology and interface designs so that they are consistent and easier to update and support. In both cases, it could be very effective to create a new system from scratch that meets these highly specialized requirements.

Custom development also allows developers to be flexible and creative in the way they solve business problems. CD Selections may envision the Web interface that takes customer orders as an important strategic enabler. The company may want to use the information from the system to better understand their customers who order over the Web, and it may want the flexibility to evolve the system to incorporate technology, such as data-mining software and geographic information systems to perform marketing research. A custom application would be easier to change to include components that take advantage of current technologies that can support such strategic efforts.

Building a system in-house also builds technical skills and functional knowledge within the company. As developers work with business users, their understanding of the business grows and they become better able to align IS with strategies and needs. These same developers climb the technology learning curve so that future projects applying similar technology become much less effortful.

Custom application development, however, also includes a dedicated effort that includes long hours and hard work. Many companies have a development staff that already is overcommitted to filling huge backlogs of systems requests, and they just do not have time for another project. Also, a variety of skills—technical, interpersonal, functional, project management, and modeling—all have to be in place for the project to move ahead smoothly. IS professionals, especially highly skilled individuals, are quite difficult to hire and retain.

The risks associated with building a system from the ground up can be quite high, and there is no guarantee that the project will succeed. Developers could be pulled away to work on other projects, technical obstacles could cause unexpected delays, and the business users could become impatient with a growing timeline.

Packaged Software

Many business needs are not unique, and because it makes little sense to “reinvent the wheel,” many organizations buy *packaged software* that has already been written, rather than developing their own custom solution. In fact, there are thousands of commercially available software programs that have already been written to serve a multitude of purposes. Think about your own need for a word processor—did you ever consider writing

CONCEPTS

IN ACTION

9-A Building a Custom System—with Some Help

I worked with a large financial institution in the southeast that suffered serious financial losses several years ago. A new CEO was brought in to change the strategy of the organization to being more “customer-focused.” The new direction was quite innovative, and it was determined that custom systems, including a data warehouse, would have to be built to support the new strategic efforts. The problem was that the company did not have the in-house skills for these kinds of custom projects.

The company now has one of the most successful data-warehouse implementations because of its willingness to use outside skills and its focus on project management. To supplement skills within the company, eight sets of external consultants, including hardware vendors, system integrators, and business strategists were hired to take part and transfer critical skills to internal employees. An in-house project manager coordinated the data-warehouse implementation full-time, and her primary goals were to set expectations clearly, define

responsibilities, and communicate the interdependencies that existed among the team members.

This company shows that successful custom development can be achieved, even when the company may not start off with the right skills in-house. But, this kind of project is not easy to pull off—it takes a talented project manager to keep the project moving along and to transition the skills to the right people over time.

—Barbara Wixom

Questions:

1. What are the risks in building a custom system without the right technical expertise? Why did the company select a project manager from within the organization? Would it have been better to hire an external professional project manager to coordinate the project?

your own word processing software? That would be very silly considering the number of good software packages available for a relatively inexpensive cost.

Similarly, most companies have needs that can be met quite well by packaged software such as payroll or accounts receivable. It can be much more efficient to buy programs that have already been created, tested, and proven, and a packaged system can be bought and installed in a relatively short period of time, when compared with a custom system. Plus, packaged systems incorporate the expertise and experience of the vendor who created the software.

Let's examine the Order Taking process once again. It turns out that there are programs available, called shopping cart programs, that allow a company to sell products on the Internet by keeping track of customers' selections, totaling them, and then e-mailing the order to a mailbox. You can easily install it on an existing Web page, and it allows you to take orders. Some shopping cart programs are even available over the Internet for free. Therefore, CD Selections could decide that acquiring a ready-made order taking application might be much more efficient than creating one from scratch.

Packaged software can range from reusable components (such as ActiveX and JavaBeans) to small single-function tools (like the shopping cart program) to huge, all-encompassing systems such as *enterprise resource planning (ERP)* applications that are installed to automate an entire business. Implementing ERP systems is a process in which large organizations spend millions of dollars installing packages by companies like SAP, PeopleSoft, Oracle, and Baan and then change their businesses accordingly. Installing ERP software is much more difficult than installing small application packages because benefits can be harder to realize and problems are much more serious.

One problem is that companies buying packaged systems must accept the functionality that is provided by the system, and rarely is there a perfect fit. If the packaged system is large in scope, its implementation could mean a substantial change in the way the company does business. Letting technology drive the business can be a dangerous way to go.

Most packaged applications allow for *customization*, or the manipulation of system parameters to change the way certain features work. For example, the package might have a way to accept information about your company or the company logo that would then appear on input screens. Or, an accounting software package could offer a choice of various ways to handle cash flow or inventory control so that it can support the accounting practices in different organizations. If the amount of customization is not enough, and the software package has a few features that don't quite work the way the company needs it to work, the project team can create workarounds.

A *workaround* is a custom-built add-on program that interfaces with the packaged application to handle special needs. It can be a nice way to create needed functionality that does not exist in the software package. But, workarounds should be a last resort for several reasons. First, workarounds are not supported by the vendor who supplied the packaged software, so when upgrades are made to the main system, they may make the workaround ineffective. Also, if problems arise, vendors have a tendency to blame the workaround as the culprit and refuse to provide support.

Although choosing a packaged software system is simpler than custom development, it too can benefit from following a formal methodology just as if you were building a custom application.

Systems integration refers to the process of building new systems by combining packaged software, existing legacy systems, and new software written to integrate these together. Many consulting firms specialize in systems integration, so it is not uncommon for companies to select the packaged software option and then outsource the integration of a variety of packages to a consulting firm. (Outsourcing is discussed in the next section.)

The key challenge in systems integration is finding ways to integrate the data produced by the different packages and legacy systems. Integration often hinges around taking data produced by one package or system and reformatting it for use in another package/system. The project team starts by examining the data produced by and needed by the different packages/systems and identifying the transformations that must occur to move the data from one to the other. In many cases, this involves "fooling" the different packages/systems into thinking that the data was produced by an existing program module that the package/system expects to produce the data rather than the new package/system that is being integrated.

For example, CD Selections needs to integrate the new Internet sales system with existing legacy systems, such as the distribution system. The distribution system was written to support a different application: the distribution of CDs to retail stores, and it currently exchanges data with the system that supports the CD Selections retail stores. The Internet sales system will need to produce data in the same format as this existing system so the distribution system will think the data is from the same system as always. Conversely, the project team may need to revise the existing distribution system, so it can accept data from the Internet sales system in a new format. A third approach is through the use of an *object wrapper*.¹¹ An object wrapper essentially is an object that "wraps around" a legacy system, enabling an object-oriented system to send messages to the legacy system. Effectively, object wrappers create an application program interface (API) to the legacy system. The creation of an object wrapper allows the protection of the corporation's investment in the legacy system.

Outsourcing

The design choice that requires the least amount of in-house resources is *outsourcing*—hiring an external vendor, developer, or service provider to create the system. Outsourcing has

¹¹ Ian Graham, *Object-Oriented Methods: Principles & Practice*, 3rd Ed. (Reading, MA: Addison-Wesley, 2001).

become quite popular in recent years. Some estimate that as many as 50 percent of companies with IT budgets of more than \$5 million are currently outsourcing or evaluating the approach. Although these figures include outsourcing for all kinds of systems functions, this section focuses on outsourcing a single development project.

There can be great benefit to having someone else develop your system. They may be more experienced in the technology or have more resources, like experienced programmers. Many companies embark upon outsourcing deals to reduce costs, while others see it as an opportunity to add value to the business. For example, instead of creating a program that handles the Order Taking Process or buying a preexisting package, CD Selections may decide to let a Web service provider provide commercial services for them.

For whatever reason, outsourcing can be a good alternative for a new system. However, it does not come without costs. If you decide to leave the creation of a new system in the hands of someone else, you could compromise confidential information or lose control over future development. In-house professionals are not benefiting from the skills that could be learned from the project, and instead the expertise is transferred to the outside organization. Ultimately, important skills can walk right out the door at the end of the contract.

Most risks can be addressed if you decide to outsource, but two are particularly important. First, assess the requirements for the project thoroughly—you should never outsource what you don't understand. If you have conducted rigorous planning and analysis, then you should be well aware of your needs. Second, carefully choose a vendor, developer, or service with a proven track record with the type of system and technology that your system needs.

Three primary types of contracts can be drawn to control the outsourcing *contract*. A *time and arrangements* deal is very flexible because you agree to pay for whatever time and expenses are needed to get the job done. Of course, this agreement could result in a large bill that exceeds initial estimates. This works best when you and the outsourcer are unclear about what it is going to take to finish the job.

You will pay no more than expected with a *fixed-price contract* because if the outsourcer exceeds the agreed-upon price, they will have to absorb the costs. Outsourcers are much more careful about defining requirements clearly up front, and there is little flexibility for change.

The type of contract gaining in popularity is the *value-added contract* whereby the outsourcer reaps some percentage of the completed system's benefits. You have very little risk in this case, but expect to share the wealth once the system is in place.

Creating fair contracts is an art because you need to carefully balance flexibility with clearly defined terms. Often needs change over time. As such, you don't want the contract to be so specific and rigid that alterations can't be made. Think about how quickly technology like the World Wide Web changes. It is difficult to foresee how a project may evolve over a long period of time. Short-term contracts help leave room for reassessment if needs change or if relationships are not working out the way both parties expected. In all cases, the relationship with the outsourcer should be viewed as a partnership where both parties benefit and communicate openly.

Managing the outsourcing relationship is a full-time job. Thus, someone needs to be assigned full-time to manage the outsourcer, and the level of that person should be appropriate for the size of the job (a multimillion dollar outsourcing engagement should be handled by a high level executive). Throughout the relationship, progress should be tracked and measured against predetermined goals. If you do embark upon an outsourcing design strategy, be sure to get more information. Many books have been written that provide much more detailed information on the topic.¹² Figure 9-10 summarizes some guidelines for outsourcing.

¹² For more information on outsourcing, we recommend Lacity, M. and Hirschheim, R., *Information Systems Outsourcing: Myths, Metaphors, and Realities* (New York, NY: Wiley, 1993) and Willcocks, L. and Fitzgerald, G., *A Business Guide to Outsourcing Information Technology* (London: Business Intelligence, 1994).

CONCEPTS

9-B EDS Value-Added Contract

IN ACTION

Value-added contracts can be quite rare—and very dramatic. They exist when a vendor is paid a percentage of revenue generated by the new system, which reduces the up-front fee, sometimes to zero. The City of Chicago and EDS (a large consulting and systems integration firm) agreed to reengineer the process by which the city collects the fines on 3.6 million parking tickets per year, and thus signed a landmark deal of this type three years ago. At the time, because of clogged courts and administrative problems, the city collected on only about 25 percent of all tickets issued. It had a \$60 million backlog of uncollected tickets.

Dallas-based EDS invested an estimated \$25 million in consulting and new systems in exchange for the right to up to 26 percent of the uncollected fines, a base

processing fee for new tickets, and software rights. To date, EDS has taken in well over \$50 million on the deal, analysts say. The deal has come under some fire from various quarters as an example of an organization giving away too much in a risk/reward-sharing deal. City officials, however, counter that the city has pulled in about \$45 million in previously uncollected fines and has improved its collection rate to 65 percent with little up-front investment.

Source: *Datamation*, February 1, 1995.

Question:

1. Do you think the City of Chicago got a good deal from this arrangement? Why or why not?

Selecting a Design Strategy

Each of the design strategies just discussed has its strengths and weaknesses, and no one strategy is inherently better than the others. Thus, it is important to understand the strengths and weaknesses of each strategy and when to use each. Figure 9-11 presents a summary of the characteristics of each strategy.

Business Need If the business need for the system is common, and technical solutions already exist in the marketplace that can meet the business need of the system, it makes little sense to build a custom application. Packaged systems are good alternatives for common business needs. A custom alternative will need to be explored when the business need is unique or has special requirements. Usually if the business need is not critical to the company, then outsourcing is the best choice—someone outside of the organization can be responsible for the application development.

In-house Experience If in-house experience exists for all of the functional and technical needs of the system, it will be easier to build a custom application than if these skills do not exist. A packaged system may be a better alternative for companies that do not have the technical skills to build the desired system. For example, a project team that does not have Web commerce technology skills may want to acquire a Web commerce package that can

Outsourcing Guidelines

- Keep the lines of communication open between you and your outsourcer.
- Define and stabilize requirements before signing a contact.
- View the outsourcing relationship as a partnership.
- Select the vendor, developer or service provider carefully.
- Assign a person to managing the relationship.
- Don't outsource what you don't understand.
- Emphasize flexible requirements, long-term relationships and short-term contracts.

FIGURE 9-10
Outsourcing Guidelines

	Use Custom Development When...	Use a Packaged System When...	Use Outsourcing When...
Business Need	The business need is unique.	The business need is common.	The business need is not core to the business.
In-house Experience	In-house functional and technical experience exists.	In-house functional experience exists.	In-house functional or technical experience does not exist.
Project Skills	There is a desire to build in-house skills.	The skills are not strategic.	The decision to outsource is a strategic decision.
Project Management	The project has a highly skilled project manager and a proven methodology.	The project has a project manager who can coordinate vendor's efforts.	The project has a highly skilled project manager at the level of the organization that matches the scope of the outsourcing deal.
Timeframe	The timeframe is flexible.	The timeframe is short.	The timeframe is short or flexible.

FIGURE 9-11 Selecting a Design Strategy

be installed without many changes. Outsourcing is a good way to bring in outside experience that is missing in-house so that skilled people are in charge of building the system.

Project Skills The skills that are applied during projects are either technical (e.g., Java, SQL) or functional (e.g., electronic commerce), and different design alternatives are more viable, depending on how important the skills are to the company's strategy. For example, if certain functional and technical expertise that relate to Internet sales applications and Web commerce application development are important to the organization because it expects the Internet to play an important role in its sales over time, then it makes sense for the company to develop Web commerce applications in-house, using company employees so that the skills can be developed and improved. On the other hand, some skills like network security may be either beyond the technical expertise of employees or not of interest to the company's strategists—it is just an operational issue that needs to be addressed. In this case, packaged systems or outsourcing should be considered so internal employees can focus on other business-critical applications and skills.

Project Management Custom applications require excellent project management and a proven methodology. There are so many things like funding obstacles, staffing hold-ups, and overly demanding business users that can push a project off-track. Therefore, the project team should choose to develop a custom application only if they are certain that the underlying coordination and control mechanisms will be in place. Packaged and outsourcing alternatives also need to be managed; however, they are more shielded from internal obstacles because the external parties have their own objectives and priorities (e.g., it may be easier for an outside contractor to say "no" to a user than a person within the company). The latter alternatives typically have their own methodologies, which can benefit companies that do not have an appropriate methodology to use.

Timeframe When time is a factor, the project team should probably start looking for a system that is already built and tested. In this way, the company will have a good idea of how long the package will take to put in place and what the final result will contain. The timeframe for custom applications is hard to pin down, especially when you consider how many projects end up missing important deadlines. If you must choose the custom development alternative, and the timeframe is very short, consider using techniques like timeboxing to manage this problem. The time to produce a system using outsourcing really

depends on the system and the outsourcer's resources. If a service provider has services in place that can be used to support the company's needs, then a business need could be implemented quickly. Otherwise, an outsourcing solution could take as long as a custom development initiative.

DEVELOPING THE ACTUAL DESIGN

Once the project team has a good understanding of how well each design strategy fits with the project's needs, they must begin to understand exactly *how* to implement these strategies. For example, what tools and technology would be used if a custom alternative were selected? What vendors make packaged systems that address the project needs? What service providers would be able to build this system if the application were outsourced? This information can be obtained from people working in the IS department and from recommendations by business users. Or, the project team can contact other companies with similar needs and investigate the types of systems that they have put in place. Vendors and consultants usually are willing to provide information about various tools and solutions in the form of brochures, product demonstrations, and information seminars. However, be sure to validate the information you receive from vendors and consultants. After all, they are trying to make a sale. Therefore, they may "stretch" the capabilities of their tool by only focusing on the positive aspects of the tool while omitting the tool's drawbacks.

It is likely that the project team will identify several ways that the system could be constructed after weighing the specific design options. For example, the project team may have found three vendors that make packaged systems that potentially could meet the project's needs. Or, the team may be debating over whether to develop a system using Java as a development tool and the database management system from Oracle; or to outsource the development effort to a consulting firm like Accenture or American Management Systems. Each alternative will have pros and cons associated with it that need to be considered, and only one solution can be selected in the end.

Alternative Matrix

An *alternative matrix* can be used to organize the pros and cons of the design alternatives so that the best solution will be chosen in the end. This matrix is created using the same steps as the feasibility analysis, which was presented in Chapter 3. The only difference is that the alternative matrix combines several feasibility analyses into one matrix so that the alternatives can be easily compared. The alternative matrix is a grid that contains the technical, budget, and organizational feasibilities for each system candidate, pros and cons associated with adopting each solution, and other information that is helpful when making comparisons. Sometimes weights are provided for different parts of the matrix to show when some criteria are more important to the final decision.

YOUR TURN

9-3 Choose a Design Strategy

Suppose that your university was interested in creating a new course registration system that can support Web-based registration. What should the university consider

when determining whether to invest in a custom, packaged, or outsourced system solution?

To create the alternative matrix, draw a grid with the alternatives across the top and different criteria (e.g., feasibilities, pros, cons, and other miscellaneous criteria) along the side. Next, fill in the grid with detailed descriptions about each alternative. This becomes a useful document for discussion because it clearly presents the alternatives being reviewed and comparable characteristics for each one.

Suppose that your company is thinking about implementing a packaged financial system like Oracle Financials or Microsoft's Great Plains, but there is not enough expertise in-house to be able to create a thorough alternative matrix. This situation is quite common—often the alternatives for a project are unfamiliar to the project team, so outside expertise is needed to provide information about the alternatives' criteria.

One helpful tool is the *request for proposals (RFP)*. An RFP is a document that solicits proposals to provide the alternative solutions from a vendor, developer, or service provider. Basically, the RFP explains the system that you are trying to build and the criteria that you will use to select a system. Vendors then respond by describing what it would mean for them to be a part of the solution. They communicate the time, cost, and exactly how their product or services will address the needs of the project.

There is no formal way to write an RFP, but it should include basic information like the description of the desired system, any special technical needs or circumstances, evaluation criteria, instructions for how to respond, and the desired schedule. An RFP can be a very large document (i.e., hundreds of pages) because companies try to include as much detail as possible about their needs so that the respondent can be just as detailed in the solution that would be provided. Thus, RFPs typically are used for large projects rather than small ones because they take a lot of time to create, and even more time and effort for vendors, developers, and service providers to develop high quality responses—only a project with a fairly large price tag would be worth the time and cost to develop a response for the RFP.

A less effort-intensive tool is a *request for information (RFI)* that includes the same format as the RFP. The RFI is shorter and contains less detailed information about a company's needs, and it requires general information from respondents that communicates the basic services that they can provide.

The final step, of course, is to decide which solution to design and implement. The decision should be made by a combination of business users and technical professionals after the issues involved with the different alternatives are well understood. Once the decision is finalized, the design phase can continue as needed, based on the selected alternative.

Applying the Concepts at CD Selections

Alec had three different approaches that he could take with the new system: he could develop the entire system using development resources from CD Selections, he could buy

YOUR TURN

9-4 Alternative Matrix

Pretend that you have been assigned the task of selecting a CASE tool for your class to use for a semester project. Using the Web or other references resources, select three CASE tools (e.g., ArgoUML, Poseidon, Rational Rose, or

Visual Paradigm). Create an alternative matrix that can be used to compare the three software products in a way in which a selection decision can be made.

a commercial Internet sales packaged software program (or a set of different packages and integrate them), or he could hire a consulting firm or service provider to create the system. Immediately, Alec ruled out the third option. Building Internet applications, especially sales systems, was important to the CD Selections' business strategy. By outsourcing the Internet sales system, CD Selections would not develop Internet application development skills and business skills within the organization.

Instead, Alec decided that a custom development project using the company's standard Web development tools would be the best choice for CD Selections. In this way, the company would be developing critical technical and business skills in-house, and the project team would be able to have a high level of flexibility and control over the final product. Also, Alec wanted the new Internet sales system to directly interface with the existing distribution system, and there was a chance that a packaged solution would not be able to integrate as well into the CD Selections environment.

There was one part of the project that potentially could be handled using packaged software: the shopping cart portion of the application. Alec realized that a multitude of programs have been written and are available (at low prices) to handle a customer's order transaction over the Web. These programs allow customers to select items for an order form, input credit card and billing information, and finalize the order transaction. Alec believed that the project team should at least consider some of these packaged alternatives so that less time had to be spent writing a program that handled basic Web tasks, and more time could be devoted to innovative marketing ideas and custom interfaces with the distribution system.

To help better understand some of the shopping cart programs that were available in the market and how their adoption could benefit the project, Alec created an alternative matrix that compared three different shopping cart programs to one another (see Figure 9-12). Although all three alternatives had positive points, Alec saw Alternative B (WebShop) as the best alternative for handling the shopping cart functionality for the new Internet sales system. WebShop was written in JAVA, the tool that CD Selections selected as its standard Web development language; the expense was reasonable, with no hidden or recurring costs; and there was a person in-house who had some positive experience with the program. Alec made a note to look into acquiring WebShop as the shopping cart program for the Internet sales system.

SUMMARY

The design phase contains many steps that guide the project team through planning out exactly how the system needs to be constructed. The requirements that were identified and the models that were created in the analysis phase serve as the primary inputs for the design activities. In object-oriented design, the primary activity is to evolve the analysis models into design models by optimizing the problem domain information already contained in the analysis models and adding system environment details to them.

Evolving the Analysis Models into Design Models

When evolving the analysis models into design models, you should first carefully review the analysis models: activity diagrams, use case descriptions, use case diagrams, CRC cards, class and object diagrams, sequence diagrams, communication diagrams, and behavioral state machines. During this review, factoring, refinement, and abstraction processes can be used to polish the current models. During this polishing, it is possible that the analysis models may become overly complex. If this occurs, then the models

	Alternative 1: Shop-With-Me	Alternative 2: WebShop	Alternative 3: Shop-N-Go
Technical Feasibility	<ul style="list-style-type: none"> Developed using C: very little C experience in-house Orders sent to company using email files 	<ul style="list-style-type: none"> Developed using C and JAVA: would like to develop in-house JAVA skills Flexible export features for passing order information to other systems 	<ul style="list-style-type: none"> Developed using JAVA: would like to develop in-house JAVA skills Orders saved to a number of file formats
Economic Feasibility	<ul style="list-style-type: none"> \$150 initial charge 	<ul style="list-style-type: none"> \$700 up front charge, no yearly fees 	<ul style="list-style-type: none"> \$200/year
Organizational Feasibility	<ul style="list-style-type: none"> Program used by other retail music companies 	<ul style="list-style-type: none"> Program used by other retail music companies 	<ul style="list-style-type: none"> Brand new application: few companies have experience with Shop-N-Go to date
Other Benefits	<ul style="list-style-type: none"> Very simple to use 	<ul style="list-style-type: none"> Tom in IS support has had limited, but positive experience with this program Easy to customize 	
Other Limitations			<ul style="list-style-type: none"> The interface is not easily customized

FIGURE 9-12 Alternative Matrix for Shopping Cart Program

should be partitioned based on the interactivity (message sending) and relationships (generalization, aggregation, and association) shared among the classes. The more a class has in common with another class (i.e., the more relationships shared), the more likely they belong in the same partition.

The second thing to do to evolve the analysis model is to add the system environment (physical architecture, user interface, and data access and management) information to the problem domain information already contained in the model. To accomplish this and to control the complexity of the models, layers are used. A layer represents an element of the software architecture of the system. We recommend five different layers to be used: foundation, physical architecture, human computer interaction, data access and management, and problem domain. Each layer only supports certain types of classes (e.g., data access manipulation classes would only be allowed on the data management layer).

Packages and Package Diagrams

A package is a general UML construct used to represent collaborations, partitions, and layers. Its primary purpose is to support the logical grouping of other UML constructs together (e.g., use cases and classes), by the developer and user to simplify and increase the understandability of a UML diagram. There are instances in which a diagram that contains only packages is useful. A package diagram contains packages and dependency relationships. A dependency relationship represents the possibility of a modification dependency existing between two packages (i.e., changes in one package could cause changes in the dependent package).

Identifying packages and creating a package diagram is accomplished using a five-step process. The five steps can be summed up as setting the context, clustering similar classes, placing the clustered classes into a package, identifying dependency relationships among the packages, and placing the dependency relationship on the package diagram.

Design Strategies

During the design phase, the project team also needs to consider three approaches to creating the new system, including developing a custom application in-house, buying a packaged system and customizing it, and relying on an external vendor, developer or system provider to build and/or support the system.

Custom development allows developers to be flexible and creative in the way they solve business problems, and it builds technical and functional knowledge within the organization. But, many companies have a development staff that is already overcommitted to filling huge backlogs of systems requests, and they just don't have time to devote to a project where a system is built from scratch. It can be much more efficient to buy programs that have been created, tested, and proven, and a packaged system can be bought and installed in a relatively short period of time, when compared with a custom solution. Workarounds can be used to meet the needs that are not addressed by the packaged application.

The third design strategy is to outsource the project and pay an external vendor, developer or service provider to create the system. It can be a good alternative for how to approach the new system; however, it does not come without costs. However, if a company decides to leave the creation of a new system in the hands of someone else, the organization could compromise confidential information or lose control over future development.

Each of the design strategies discussed above has its strengths and weaknesses, and no one strategy is inherently better than the others. Thus, it is important to consider such issues as the uniqueness of business need for the system, the amount of in-house experience that is available to build the system, and the importance of the project skills to the company. Also, the existence of good project management and the amount of time available to develop the application play a role in the selection process.

Developing the Actual Design

Ultimately, the decision must be made regarding the specific type of system that needs to be designed. An alternative matrix can help make this decision by presenting feasibility information for several candidate solutions in a way in which they can be compared easily. Both the Request for Proposal and Request for Information are two ways to gather accurate information regarding the alternatives.

KEY TERMS

A-Kind-Of	Enterprise resource systems (ERP)	Package
Abstract classes	Factoring	Package diagram
Abstraction	Fixed-price contract	Packaged software
Aggregation	Foundation layer	Partition
Alternative matrix	Generalization	Physical architecture layer
Class	Has-Parts	Problem domain layer
Client	Human computer interaction layer	Refinement
Collaboration	Layer	Request for information
Concrete classes	Message	Request for proposals
Contract	Method	Server
Controller	Middleware	Smalltalk
Custom development	Model	Systems integration
Customization	Model-View-Controller (MVC)	Time and arrangements contract
Data management layer	Module	Value-added contract
Dependency relationship	Object wrapper	View
Design phase	Outsourcing	Workaround

QUESTIONS

1. What is the primary difference between an analysis model and a design model?
2. What does factoring mean? How is it related to abstraction and refinement?
3. What is a partition? How does a partition relate to a collaboration?
4. What is a layer? Name the different layers.
5. What is a package? How are packages related to partitions and layers?
6. What is a dependency relationship? How do you identify them?
7. What are the five steps for identifying packages and creating package diagrams?
8. What situations are most appropriate for a custom development design strategy?
9. What are some problems with using a packaged software approach to building a new system? How can these problems be addressed?
10. Why do companies invest in ERP systems?
11. What are the pros and cons of using a workaround?
12. When is outsourcing considered a good design strategy? When is it not appropriate?
13. What are the differences between the time and arrangements, fixed-price, and value-added contracts for outsourcing?
14. How are the alternative matrix and feasibility analysis related?
15. What is an RFP? How is this different from an RFI?

EXERCISES

- A. Based on the functional models you created for exercises E, F, and G in Chapter 6 and the structural model you created in Exercise K in Chapter 7, draw a package diagram for the problem domain layer for the dentist office system.
- B. Based on the functional models you created for exercises J and K in Chapter 6 and the structural model you created in Exercise M in Chapter 7, draw a package diagram for the problem domain layer for the real estate system.
- C. Based on the functional models you created for exercises L and M in Chapter 6, the structural model you created in Exercise N in Chapter 7, and the behavioral models you created in exercises A and B in Chapter 8, draw a package diagram for the problem domain layer for the video store system.
- D. Based on the functional models you created for exercises N and O in Chapter 6, the structural model you created in Exercise O in Chapter 7, and the behavioral models you created in exercises C and D in Chapter 8, draw a package diagram for the problem domain layer for the health club membership system.
- E. Based on the functional models you created for exercises P and Q in Chapter 6, the structural model you created in Exercise P in Chapter 7, and the behavioral models you created in exercises H in Chapter 8, draw a package diagram for the problem domain layer for the catering system.
- F. Based on the functional models you created for exercises R and S in Chapter 6, the structural model you created in Exercise Q in Chapter 7, and the behavioral models you created in exercises I in Chapter 8, draw a package diagram for the problem domain layer for the Of-the-Month Club.
- G. Based on the functional models you created for exercises T and U in Chapter 6, the structural model you created in Exercise R in Chapter 7, and the behavioral models you created in exercises J in Chapter 8, draw a package diagram for the problem domain layer for the university library system.
- H. Which design strategy would you recommend for the construction of this system in:
 - a. Exercise A. Why?
 - b. Exercise B. Why?
 - c. Exercise C. Why?
 - d. Exercise D. Why?
 - e. Exercise E. Why?
 - f. Exercise F. Why?
 - g. Exercise G. Why?
- I. Pretend that you are leading a project that will implement a new course enrollment system for your university. You are thinking about either using a packaged course enrollment application or outsourcing the job to an external consultant. Create an outline for a Request for Proposal to which interested vendors and consultants could respond.
- J. Pretend that you and your friends are starting a small business painting houses in the summertime. You need to buy a software package that handles the financial transactions of the business. Create an alternative matrix that compares three packaged systems (e.g., Quicken, MS Money, Quickbooks). Which alternative appears to be the best choice?

MINICASES

- Susan, president of MOTO, Inc., a human resources management firm, is reflecting on the client management software system her organization purchased four years ago. At that time, the firm had just gone through a major growth spurt, and the mixture of automated and manual procedures that had been used to manage client accounts became unwieldy. Susan and Nancy, her IS department head, researched and selected the package that is currently used. Susan had heard about the software at a professional conference she attended, and at least initially, it worked fairly well for the firm. Some of their procedures had to change to fit the package, but they expected that and were prepared for it.

Since that time, MOTO, Inc. has continued to grow, not only through an expansion of the client base, but also through the acquisition of several smaller employment-related businesses. MOTO, Inc. is a much different business than it was four years ago. Along with expanding to offer more diversified human resource management services, the firm's support staff has also expanded. Susan and Nancy are particularly proud of the IS department they have built up over the years. Using strong ties with a local university, an attractive compensation package, and a good working environment, the IS department is well-staffed with competent, innovative people, plus a steady stream of college interns that keeps the department fresh and lively. One of the IS teams pioneered the use of the Internet to offer MOTO's services to a whole new market segment, an experiment that has proven very successful.

It seems clear that a major change is needed in the client management software, and Susan has already begun to plan financially to undertake such a project. This software is a central part of MOTO's operations, and Susan wants to be sure that a quality system is obtained this time. She knows that the vendor of their current system has made some revisions and additions to its product line. There are also a number of other software vendors who offer products that may be suitable. Some of these vendors did not exist when the purchase was made four years ago. Susan is also

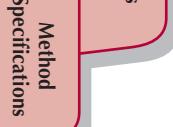
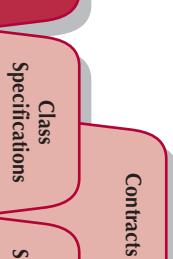
considering Nancy's suggestion that the IS department develop a custom software application.

- Outline the issues that Susan should consider that would support the development of a custom software application in-house.
 - Outline the issues that Susan should consider which would support the purchase of a software package.
 - Within the context of a systems development project, when should the decision of "make-versus-buy" be made? How should Susan proceed? Explain your answer.
- Refer to Minicase 1 in Chapter 7. After completing all of the analysis models (both the As-Is and To-Be models) for West Star Marinas, the Director of Operations finally understood why it was important to understand the As-Is system before delving into the development of the To-Be system. However, you now tell him that the To-Be models are only the problem domain portion of the design. To say the least, he is now very confused. After explaining to him the advantages of using a layered approach to developing the system, he informs you that "I don't care about reusability or maintenance. I only want the system to be implemented as soon as possible. You IS types are always trying to pull a fast one on the users. Just get the system completed."

What is your response to the Director of Operations? Do you jump into implementation as he seems to want? What do you do next?
 - Refer to the analysis models that you created for Professional and Scientific Staff Management (PSSM) for Minicase 2 in Chapter 6 and for Minicase 1 in Chapter 8.
 - Add packages to your use-case diagram to simplify it.
 - Use the communication diagram to identify logical partitions in your class diagram. Add packages to the diagram to represent the partitions.
 - Refer to the analysis models that you created for Holiday Travel Vehicles for Minicase 2 in Chapter 7 and for Minicase 2 in Chapter 8.
 - Add packages to your use-case diagram to simplify it.
 - Use the communication diagram to identify logical partitions in your class diagram. Add packages to the diagram to represent the partitions.

CHAPTER 9 ■

MOVING ON
TO DESIGN



PART THREE

DESIGN PHASE

The Design Phase decides *how* the system will operate. First, the project team creates a Design Plan and a set of factored and partitioned Analysis models (Functional, Structural, and Behavioral). The class and method designs are illustrated using the Class Specifications (using CRC Cards and Class Diagrams), Contracts, and Method Specifications. Next, the data management layer is addressed by designing the actual database or file structure to be used for object persistence and a set of classes that will map the Class Specifications into the object persistence format chosen. Next, the team produces the user interface layer design using Use Scenarios, Windows Navigation Diagrams, Real Use Cases, Interface Standards, and User Interface Templates. The physical architecture layer design is created using a Deployment Diagrams and Hardware/Software Specification. This collection of deliverables is the system specification that is handed to the programming team for implementation. At the end of the Design Phase, the feasibility analysis and project plan are reexamined and revised, and another decision is made by the project sponsor and approval committee about whether to terminate the project or continue.

CHAPTER 10 ■

CLASS AND
METHOD DESIGN

CHAPTER 11 ■

DATA
MANAGEMENT
LAYER DESIGN

CHAPTER 12 ■

HUMAN COMPUTER
INTERACTION
LAYER DESIGN

CHAPTER 13 ■

PHYSICAL
ARCHITECTURE
LAYER DESIGN

CHAPTER 9

MOVING ON TO DESIGN

The design phase in object-oriented system development uses the requirements that were gathered during analysis to create a blueprint for the future system. A successful design builds upon what was learned in earlier phases and leads to a smooth implementation by creating a clear, accurate plan of what needs to be done. This chapter describes the initial transition from analysis to design and presents three ways to approach the design for the new system.

OBJECTIVES

- Understand the transition from analysis to design.
- Understand the use of factoring, partitions, and layers.
- Be able to create package diagrams.
- Be familiar with the custom, packaged, and outsource design alternatives.
- Be able to create an alternative matrix.

CHAPTER OUTLINE

Introduction	Design Strategies
Evolving the Analysis Models into Design Models	Custom Development
Factoring	Packaged Software
Partitions and Collaborations	Outsourcing
Layers	Selecting a Design Strategy
Packages and Package Diagrams	Developing the Actual Design
Identifying Packages and Creating Package Diagrams	Alternative Matrix
Applying the Concepts at CD Selections	Applying the Concepts at CD Selections
	Summary

INTRODUCTION

The purpose of the analysis phase is to figure out *what* the business needs. The purpose of the *design phase* is to decide *how* to build it. The major activity that takes place during the design phase is evolving the set of analysis representations into design representations.

Throughout the design phase, the project team carefully considers the new system in respect to the current environment and systems that exist within the organization as a whole. Major considerations of the “*how*” of a system are environmental factors like integrating with existing systems, converting data from legacy systems, and leveraging skills that exist in-house. Although the Planning and Analysis phases are undertaken to develop

a “possible” system, the goal of the Design phase is to create a blueprint for a system that makes sense to implement.

An important initial part of the design phase is to examine several design strategies and decide which will be used to build the system. Systems can be built from scratch, purchased and customized, or outsourced to others, and the project team needs to investigate the viability of each alternative. The decision to make, to buy, or to outsource influences the design tasks that are accomplished throughout the rest of the phase.

At the same time, detailed design of the individual classes and methods that are used to map out the nuts and bolts of the system and how they are to be stored must still be completed. Techniques like CRC cards, class diagrams, contract specification, method specification, and database design provide detail in preparation for the implementation phase, and they ensure that programmers have sufficient information to build the right system efficiently. These topics are covered in Chapters 10 and 11.

Design also includes activities like designing the user interface, system inputs, and system outputs, which involve the ways that the user interacts with the system. Chapter 12 describes these three activities in detail, along with techniques, such as story boarding and prototyping that help the project team design a system that meets the needs of its users and is satisfying to use.

Finally, physical architecture decisions are made regarding the hardware and software that will be purchased to support the new system and the way that the processing of the system will be organized. For example, the system can be organized so that its processing is centralized at one location, distributed, or both centralized and distributed, and each solution offers unique benefits and challenges to the project team. Global issues and security need to be considered along with the system’s technical architecture because they will influence the implementation plans that are made. Physical architecture, security, and global issues will be described in Chapter 13.

The many steps of the design phase are highly interrelated and, as with the steps in the analysis phase, the analysts often go back and forth among them. For example, prototyping in the interface design step often uncovers additional information that is needed in the system. Alternatively, a system that is being designed for an organization that has centralized systems may require substantial hardware and software investments if the project team decides to change to a system in which all of the processing is distributed.

In this chapter, we overview the processes that are used to evolve the analysis models into design models. Next, we introduce the use of packages and package diagrams. Finally, we examine the three fundamental approaches to developing new systems: make, buy, or outsource.

EVOLVING THE ANALYSIS MODELS INTO DESIGN MODELS

The purpose of the analysis models was to represent the underlying business problem domain as a set of collaborating objects. In other words, the analysis activities defined the functional requirements. To achieve this, the analysis activities ignored nonfunctional requirements such as performance and the system environment issues (e.g., distributed or centralized processing, user interface issues, and database issues). In contrast, the primary purpose of the design models is to increase the likelihood of successfully delivering a system that implements the functional requirements in a manner that is affordable and easily maintainable.

Therefore, in system design, we address both the functional and nonfunctional requirements. From an object-oriented perspective, system design models simply refine the system analysis models by adding system environment (or solution domain) details to them and refining the problem domain information already contained in the analysis models.

PRACTICAL**TIP****Avoiding Classic Design Mistakes**

In Chapters 3 and 4, we discussed several classic mistakes and how to avoid them. Here, we summarize four classic mistakes in the design phase, and discuss how to avoid them.

1. Reducing design time: If time is short, there is a temptation to reduce the time spent in “unproductive” activities such as design so that the team can jump into “productive” programming. This results in missing important details that have to be investigated later at a much higher time cost (usually at least ten times longer).

Solution: If time pressure is intense, use timeboxing to eliminate functionality or move it into future versions.

2. Feature creep: Even if you are successful at avoiding scope creep, about 25 percent of system requirements will still change. And, changes—big and small—can significantly increase time and cost.

Solution: Ensure that all changes are vital and that the users are aware of the impact on cost and time. Try to move proposed changes into future versions.

3. Silver bullet syndrome: Analysts sometimes believe the marketing claims for some design tools that claim to solve all problems and magically reduce time and costs. No one tool or technique can eliminate overall time or costs by more than 25 percent (although some can reduce individual steps by this much).

Solution: If a design tool has claims that appear too good to be true, just say no.

4. Switching tools in mid-project: Sometimes analysts switch to what appears to be a better tool during design in the hopes of saving time or costs. Usually, any benefits are outweighed by the need to learn the new tool. This also applies to even “minor” upgrades to current tools.

Solution: Don’t switch or upgrade unless there is a compelling need for specific features in the new tool, and then explicitly increase the schedule to include learning time.

Adapted from Steve McConnell, *Rapid Development*, Microsoft Press, Redmond, WA 1996.

When evolving the analysis model into the design model, you should first carefully review the use cases and the current set of classes (their methods and attributes, and the relationships between them). Are all of the classes necessary? Are there any missing classes? Are the classes fully defined? Are there any missing attributes or methods? Do the classes have any unnecessary attributes and methods? Is the current representation of the evolving system optimal?

In the following sections, we introduce factoring, partitions and collaborations, and layers as a way to evolve problem domain-oriented analysis models into optimal solution domain-oriented design models.

Factoring

Factoring is the process of separating out a *module* into a standalone module in and of itself. The new module can be a new *class* or a new *method*. For example, when reviewing a set of classes, it may be discovered that they have a similar set of attributes and methods. As such, it may make sense to factor out the similarities into a separate class. Depending on whether the new class should be in a superclass relationship to the existing classes or not, the new class can be related to the existing classes through *generalization (A-Kind-Of)* or possibly through *aggregation (Has-Parts)* relationship. For example, using the appointment system example in the previous chapters (see Figure 7-2), if the Employee class had not been identified, we could possibly identify it at this stage by factoring out the similar methods and attributes from the Nurse, Administrative Staff, and Doctor classes.

In this case, we would relate the new class (Employee) to the existing classes using the generalization (A-Kind-Of) relationship.

Abstraction and *refinement* are two closely related processes to factoring. Abstraction deals with the creation of a “higher” level idea from a set of ideas. Identifying the Employee class is an example of abstracting from a set of lower classes to a higher one. In some cases, the abstraction process will identify *abstract classes*, whereas, in other situations, it will identify additional *concrete classes*.¹ The refinement process is the opposite of the abstraction process. In the appointment system example in the previous chapters (see Figure 7-2), we could identify additional subclasses of the Administrative Staff class, such as Receptionist, Secretary, and Bookkeeper. Of course we would only add the new classes if there were sufficient differences between them. Otherwise, the more general class, Administrative Staff, would suffice.

Partitions and Collaborations

Based on all of the factoring, refining, and abstracting that can take place to the evolving system, the sheer size of the system representation can overload both the user and the developer. At this point in the evolution of the system, it may make sense to split the representation into a set of *partitions*. A partition is the object-oriented equivalent of a subsystem,² where a subsystem is a decomposition of a larger system into its component systems (e.g., an accounting information system could be functionally decomposed into an accounts payable system, an account receivable system, a payroll system, and so on). From an object-oriented perspective, partitions are based on the pattern of activity (messages sent) among the objects in an object-oriented system.

A good place to look for potential partitions is the *collaborations* modeled in UML’s communication diagrams (see Chapter 8). If you recall, one useful way to identify collaborations is to create a communication diagram for each use case. However, since an individual class may support multiple use cases, an individual class can participate in multiple use case-based collaborations. In those cases where classes are supporting multiple use cases, the collaborations should be merged together. Also, CRUD analysis (see Chapter 8) can be used to identify potential classes on which to merge collaborations.

Depending on the complexity of the merged collaboration, it may be useful in decomposing the collaboration into multiple partitions. In this case, in addition to having collaborations between objects, it is possible to have collaborations among partitions. The general rule of thumb is the more messages sent between objects, the more likely the objects belong in the same partition. The fewer messages sent, the less likely the two objects belong together.

Another useful approach to identify potential partitions is to model each collaboration between objects in terms of clients, servers, and contracts. A *client* is an instance of a class that sends a *message* to an instance of another class for a *method* to be executed; a *server* is the instance of a class that receives the message; and a *contract* is the specification that formalizes the interactions between the client and server objects (see Chapters 7 and 10). This approach allows the developer to build up potential partitions by looking at the contracts that have been specified between objects. In this case, the more contracts there are between objects, the more often than not the objects belong in the same partition. The fewer contracts, the chances are the two classes do not belong in the same partition.

¹ See Chapter 7 for the differences between abstract and concrete classes.

² Some authors refer to partitions as subsystems (e.g., see Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Weiner, *Designing Object-Oriented Software* [Englewood Cliffs, NJ: Prentice-Hall, 1990]), while others refer to them as layers (e.g., see Ian Graham, *Migrating to Object Technology* [Reading, MA Addison-Wesley, 1994]). However, we have chosen to use the term partition (from Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design* [Englewood Cliffs, NJ: Prentice Hall, 1998]) to minimize confusion between subsystems in a traditional system development approach and the layers associated with Rational’s Unified Approach.

**YOUR
TURN**

9-1 Campus Housing

You have been working with the system for the campus housing service over the previous three chapters. In Chapter 6, you were asked to create a set of use cases (Your Turn 6-6) and to create a use case diagram (Your Turn 6-8). In Chapter 7 (Your Turn 7-3), you created a structural model (CRC cards and class diagram) for the same situation. In Chapter 8, you created a sequence diagram (Your Turn 8-1) and a communication diagram (Your Turn 8-2) for one of

the use cases you had identified in Chapter 6. Finally, you also created a behavioral state machine for the apartment class in Chapter 8 (Your Turn 8-3).

Based on the current set of functional, structural, and behavioral models portrayed in these diagrams, apply the abstraction and refinement processes to identify additional abstract and concrete classes that would be useful to include in the evolving system.

Layers

Until this point in the development of our system, we have focused only on the problem domain; we have totally ignored the system environment (physical architecture, user interface, and data access and management). To successfully evolve the analysis model of the system into a design model of the system, we must add the system environment information. One useful way to do this, without overloading the developer, is to use *layers*. A layer represents an element of the software architecture of the evolving system. We have focused only on one layer in the evolving software architecture: the problem domain layer. There should be a layer for each of the different elements of the system environment (e.g., system architecture, user interface, and data access and management).

The idea of separating the different elements of the architecture into separate layers can be traced back to the MVC architecture of *Smalltalk*.³ When Smalltalk was first created,⁴ the authors decided to separate the application logic from the logic of the user interface. In this manner, it was possible to easily develop different user interfaces that worked with the same application. To accomplish this, they created the *Model-View-Controller* (MVC) architecture where *Models* implemented the application logic (problem domain), and *Views* and *Controllers* implemented the logic for the user interface. Views handled the output and Controllers handled the input. Since graphical user interfaces were first developed in the Smalltalk language, the MVC architecture served as the foundation for virtually all graphical user interfaces that have been developed today (including the Mac interfaces, the Windows family, and the various Unix-based GUI environments).

Based on Smalltalk's innovative MVC architecture, many different software layers have been proposed.⁵ Based on these proposals, we suggest the following layers on which to base

³ See Simon Lewis, *The Art and Science of Smalltalk: An Introduction to Object-Oriented Programming Using VisualWorks* (Englewood Cliffs, NJ: Prentice-Hall, 1995).

⁴ Smalltalk was first invented in the early 1970s by a software development research team at Xerox PARC. It introduced many new ideas into the area of programming languages (e.g., object-orientation, windows-based user interfaces, reusable class library, and the idea of a development environment). In many ways, Smalltalk is the father (or mother) of all object-based and object-oriented languages, such as Visual Basic, C++, and Java.

⁵ For example, Problem Domain, Human Interaction, Task Management, and Data Management (Peter Coad and Edward Yourdon, *Object-Oriented Design* [Englewood Cliffs, NJ: Yourdon Press, 1991]); Domain, Application, and Interface (Ian Graham, *Migrating to Object Technology* [Reading, MA: Addison Wesley, 1994]); Domain, Service, and Presentation (Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design* [Englewood Cliffs, NJ: Prentice Hall, 1998]); Business, View, and Access (Ali Bahrami, *Object-Oriented Systems Development using the Unified Modeling Language*, [New York, NY: McGraw Hill, 1999]); Application-Specific, Application-General, Middleware, System-Software (Ivar Jacobson, Grady Booch, and James Rumbaugh, *The Unified Software Development Process* [Reading, MA: Addison-Wesley, 1999]); and Foundation, Architecture, Business, and Application (Meilir Page-Jones, *Fundamentals of Object-Oriented Design in UML* [Reading, MA: Addison-Wesley, 2000]).

software architecture: foundation, physical architecture, human computer interaction, data access and management, and problem domain (see Figure 9-1). Each layer limits the types of classes that can exist on it (e.g., only user interface classes may exist on the human computer interaction layer). The following provides a short description of each layer.

Foundation The *foundation layer* is, in many ways, a very uninteresting layer. It contains classes that are necessary for any object-oriented application to exist. They include classes that represent fundamental data types (e.g., integers, real numbers, characters, and strings), classes that represent fundamental data structures (sometimes referred to as container classes; e.g., lists, trees, graphs, sets, stacks, and queues), and classes that represent useful abstractions (sometimes referred to as utility classes; e.g., date, time, and money). Today, the classes found on this layer typically are included with the object-oriented development environments.

Physical Architecture The *physical architecture layer* addresses how the software will execute on specific computers and networks. As such, this layer includes classes that deal with communication between the software and the computer's operating system and the network. For example, classes that address how to interact with the various ports on a specific computer would be included in this layer. This layer also includes classes that would interact with so-called *middleware* applications, such as the OMG's CORBA and Microsoft's DCOM architectures that handle distributed objects.

Unlike the foundation layer, there are many design issues that must be addressed before choosing the appropriate set of classes for this layer. These design issues include the choice of a computing or network architecture (such as the various client-server architectures), the actual design of a network, hardware and server software specification, global/international issues (such as multilingual requirements), and security issues. A complete description of all of the issues related to system architecture is beyond the scope of this book—there are entire courses dedicated to this subject. However, we do present the basic issues in Chapter 13.

Human Computer Interaction The *human computer interaction layer* contains classes associated with the View and Controller idea from Smalltalk. The primary purpose of this layer is to keep the specific user interface implementation separate from the problem domain classes. This increases the portability of the evolving system. Typical classes found on this layer include classes that can be used to represent buttons, windows, text fields, scroll bars, check boxes, drop-down lists, and many other classes that represent user interface elements.

When it comes to designing the user interface for an application, there are many issues that must be addressed. For example, how important is consistency across different user interfaces, what about differing levels of user experience, how is the user expected to be able to navigate through the system, what about help systems and online manuals, what types

Layers	Examples	Relevant Chapters
Foundation	Date, Enumeration	9, 10
Physical Architecture	ServerSocket, URLConnection	10, 13
Human Computer Interaction	Button, Panel	10, 12
Data Management	DataInputStream, FileInputStream	10, 11
Problem Domain	Employee, Customer	6, 7, 8, 9, 10

FIGURE 9-1
Layers and Sample
Classes

of input elements should be included (e.g., text box, radio buttons, check boxes, sliders, drop-down list boxes, etc.), and what types of output elements should be included (e.g., text, tables, graphs, etc.). Like the physical architecture layer, a complete description of all of the issues related to human computer interaction is beyond the scope of this book.⁶ However from the user's perspective, the user interface is the system. As such, we present the basic issues in user interface design in Chapter 12.

Data Management The *data management layer* addresses the issues involving the persistence of the objects contained in the system. The types of classes that appear in this layer deal with how objects can be stored and retrieved. Like the human computer interface layer, the classes contained in this layer allow the problem domain classes to be independent of the storage utilized, and hence increase the portability of the evolving system. Some of the issues related to this layer include choice of the storage format (such as relational, object/relational, and object databases) and storage optimization (such as clustering and indexing). A complete description of all of the issues related to the data management layer also is beyond the scope of this book.⁷ However, we do present the fundamentals in Chapter 11.

Problem Domain The previous layers dealt with classes that represent elements from the system environment and which will be added to the evolving system specification. The *problem domain layer* is what we have focused our attention on up until now. At this stage of the development of our system, we will need to further detail the classes so that it will be possible to implement them in an effective and efficient manner.

Many issues need to be addressed when designing classes, no matter on which layer they appear. For example, there are issues related to factoring, cohesion and coupling, concreteness, encapsulation, proper use of inheritance and polymorphism, constraints, contract specification, and detailed method design. These issues are discussed in Chapter 10.

PACKAGES AND PACKAGE DIAGRAMS

In UML, collaborations, partitions, and layers can be represented by a higher-level construct: a package.⁸ A *package* is a general construct that can be applied to any of the elements in UML models. In Chapter 6, we introduced the idea of packages as a way to group use cases together to make the use case diagrams easier to read and to keep the models at a reasonable level of complexity. In Chapters 7 and 8, we did the same thing for class and communication diagrams, respectively. In this section we describe a package diagram: a diagram that is composed only of packages. A *package diagram* is effectively a class diagram that only shows packages.

The symbol for a package is similar to a tabbed folder (see Figure 9-2). Depending on where a package is used, packages can participate in different types of relationships. For example, in a class diagram, packages represent groupings of classes. Therefore, aggregation and association relationships are possible.

⁶ One of the best books on user interface design is Ben Schneiderman, *Designing the User Interface: Strategies for Effective Human Computer Interaction*, 3rd Ed (Reading, MA: Addison-Wesley, 1998).

⁷ There are many good database design books that are relevant to this layer, see for example, Fred R. McFadden, Jeffrey A. Hoffer, Mary B. Prescott, *Modern Database Management*, 4th ed. (Reading, MA: Addison-Wesley, 1998), Michael Blaha and William Premerlani, *Object-Oriented Modeling and Design for Database Applications* (Englewood Cliffs, NJ: Prentice Hall, 1998), and Robert J. Muller, *Database Design for Smarties: Using UML for Data Modeling* (San Francisco, CA: Morgan Kaufmann, 1999).

⁸ This discussion is based on material in Chapter 7 of Martin Fowler with Kendall Scott, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 2nd Ed. (Reading, MA: Addison Wesley, 2000).

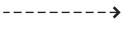
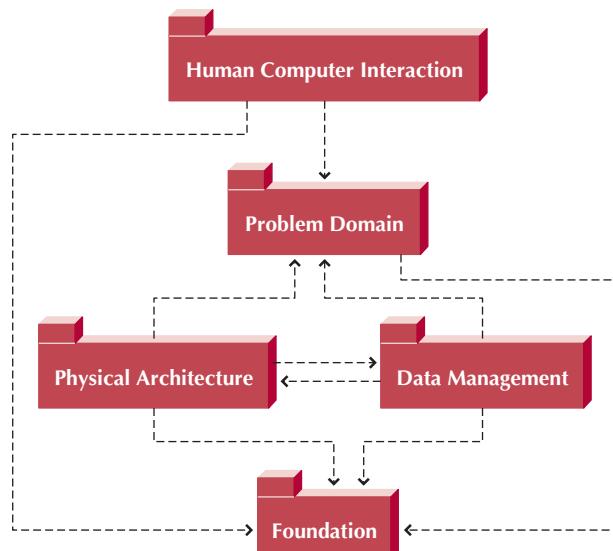
<p>A Package:</p> <ul style="list-style-type: none"> ■ A logical grouping of UML elements ■ Used to simplify UML diagrams by grouping related elements into a single higher-level element 	 Package
<p>A Dependency Relationship:</p> <ul style="list-style-type: none"> ■ Represents a dependency between packages, i.e., if a package is changed, the dependent package also could have to be modified ■ The arrow is drawn from the dependent package toward the package on which it is dependent 	

FIGURE 9-2 Syntax for Package Diagram

In a package diagram, a new relationship, the *dependency relationship*, is useful to depict. A dependency relationship is portrayed by a dashed arrow (see Figure 9-2). A dependency relationship represents the fact that a modification dependency exists between two packages. That is, it is possible that a change in one package potentially could cause a change to be required in another package. Figure 9-3 portrays the dependencies among the different layers (foundation, physical architecture, human computer interaction, data access and management, and problem domain). For example, if a change occurs in the problem domain layer, it most likely will cause changes to occur in the human computer interaction, physical architecture, and data management layers. Notice that these layers “point to” the problem domain layer; as such, they are dependent on it. However, the reverse is not true.

At the class level, there could be many causes for dependencies among classes. For example, if the protocol for a method is changed, then this causes the interface for all objects of this class to change. Therefore, all classes that have objects that send messages to the instances of the modified class may have to be modified. Capturing dependency relationships among the classes and packages helps the organization in maintaining object-oriented information systems.

**FIGURE 9-3**
Package Diagram of
Dependency Relation-
ships among Layers

As already stated, collaborations, partitions, and layers are modeled as packages in UML. Furthermore, collaborations are normally factored into a set of partitions, which are typically placed on a layer. In addition, partitions can be composed of other partitions. Also, it is possible to have classes in partitions, which are contained in another partition, which is placed on a layer. All of these groupings are represented using packages in UML. Remember that a package is simply a generic, grouping construct used to simplify UML models through the use of composition.⁹

A simple package diagram, based on the appointment system example from the previous chapters, is shown in Figure 9-4. This diagram only portrays a very small portion of the entire system. In this case, we see that the Patient UI, DAM-Patient, and Patient Table classes are dependent on the Patient class. Furthermore, the DAM-Patient class is dependent on the Patient Table class. The same can be seen with the classes dealing with the actual appointments. By isolating the Problem Domain classes (such as the Patient and Appt classes) from the actual object persistence classes (such as the Patient Table and Appt Table classes) through the use of the intermediate Data Access and Manipulation classes (DAM-Patient and DAM-Appt classes), we isolate the Problem Domain classes from the actual storage medium.¹⁰ This greatly simplifies the maintenance and increases the reusability of the Problem Domain classes. Of course, in a complete description of a real system, there would be many more dependencies.

Identifying Packages and Creating Package Diagrams

In this section, we describe a simple five-step process to create package diagrams (see Figure 9-5). The first step is to set the context for the package diagram. Remember, packages can be used to model partitions and/or layers. Revisiting the appointment system again, let's set the context as the problem domain layer.

The second step is to cluster the classes together into partitions based on the relationships that the classes share. The relationships include generalization, aggregation, the various associations, and the message sending that takes place between the objects in the system. To identify the packages in the appointment system, we should look at the different analysis models (e.g., the class diagram [see Figure 7-2], sequence diagrams [see Figure 8-1], and the communication diagrams [see Figure 8-6]). Any classes in a generalization hierarchy should be kept together in a single partition.

The third step is to place the clustered classes together in a partition and model the partitions as packages. Figure 9-6 portrays five packages: PD Layer, Person Pkg, Patient Pkg, Appt Pkg, and Treatment Pkg.

The fourth step is to identify the dependency relationships among the packages. In this case, we review the relationships that cross the boundaries of the packages to uncover potential dependencies. In the appointment system, we see association relationships that connect the Person Pkg with the Appt Pkg (via the association between the Doctor class and the Appointment class), and the Patient Pkg, which is contained within the Person Pkg, with the Appt Pkg (via the association between the Patient and Appointment classes) and the Treatment Pkg (via the association between the Patient and Symptom classes).

The fifth step is to place the dependency relationships on the evolved package diagram. In the case of the Appointment system, there are dependency relationships between the Person Pkg and the Appt Pkg and the Person Pkg and the Treatment Pkg. To increase the

⁹ For those familiar with traditional approaches, such as structured analysis and design, packages serve a similar purpose as the leveling and balancing processes used in data flow diagramming.

¹⁰ These issues are described in more detail in Chapter 11.

understandability of the dependency relationships among the different packages, a pure package diagram that only shows the dependency relationships among the packages can be created (see Figure 9-7).

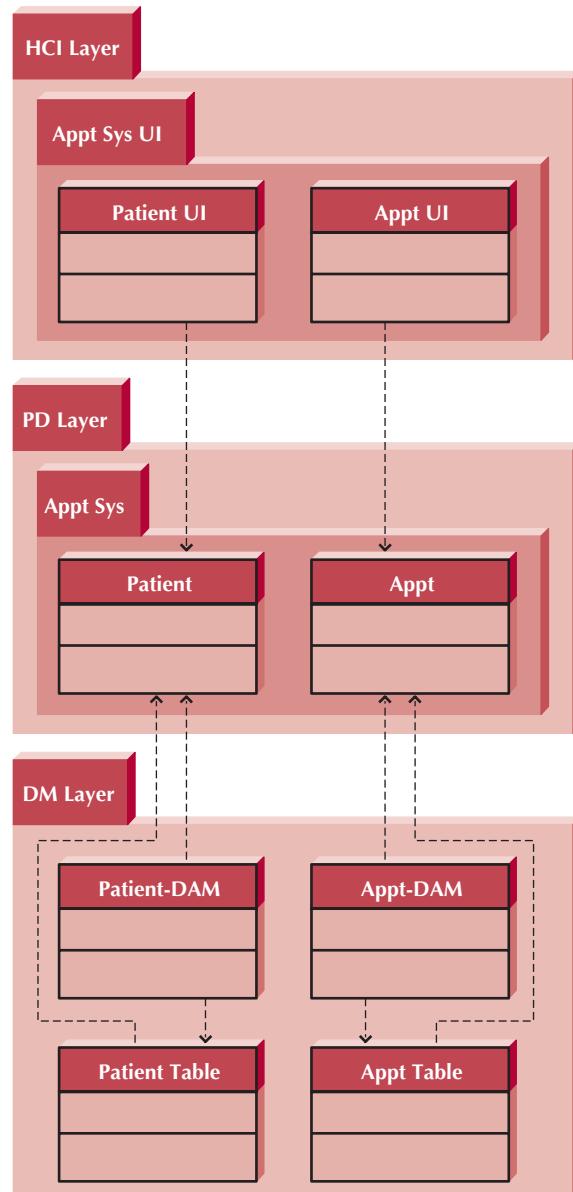


FIGURE 9-4
Partial Package Diagram of the Appointment System

FIGURE 9-5 Steps for Identifying Packages and Building Package Diagram

1. Set the Context.
2. Cluster classes together based on shared relationships.
3. Model clustered classes as a package.
4. Identify dependency relationships among packages.
5. Place dependency relationships between packages.

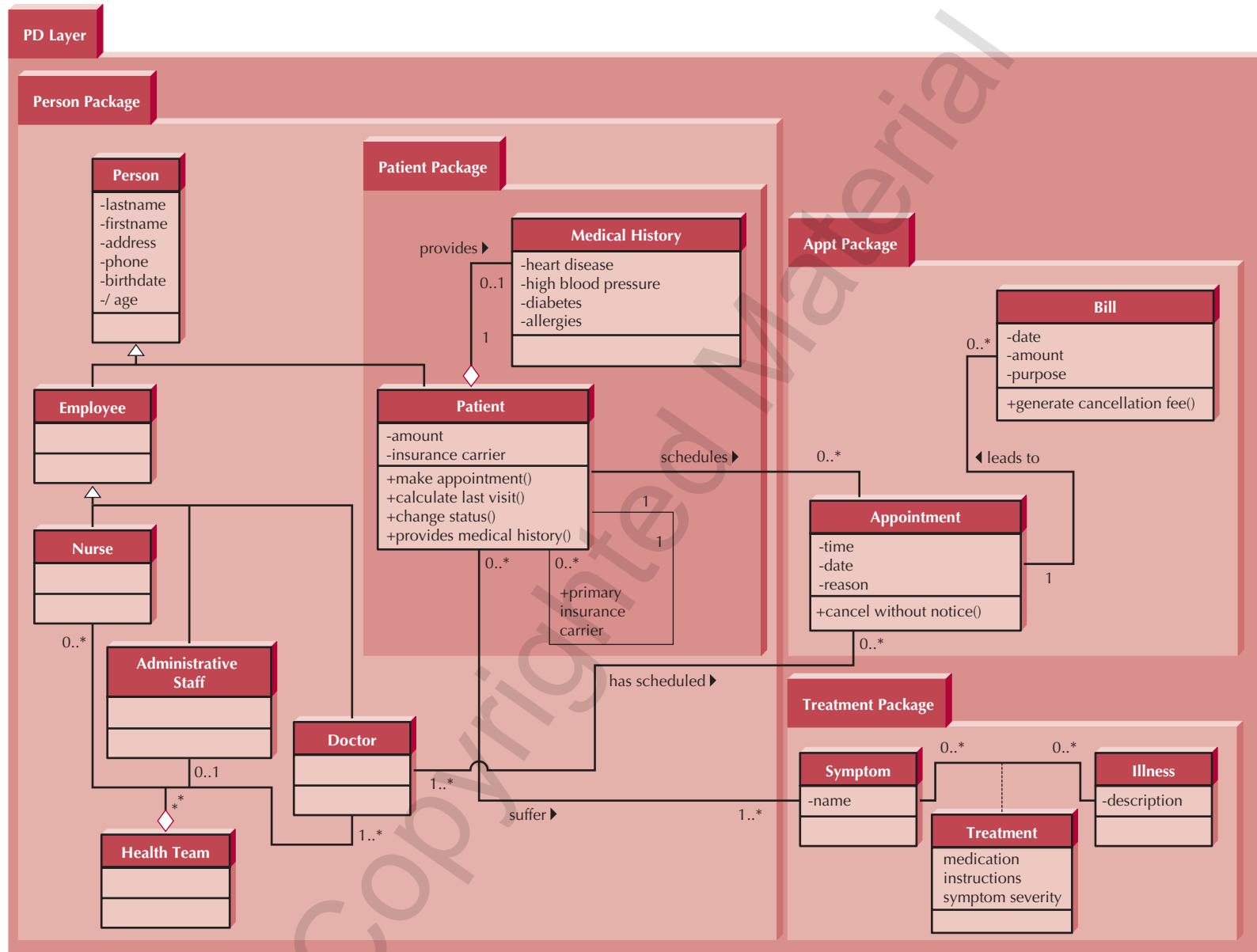


FIGURE 9-6 Package Diagram of the PD Layer for the Appointment System

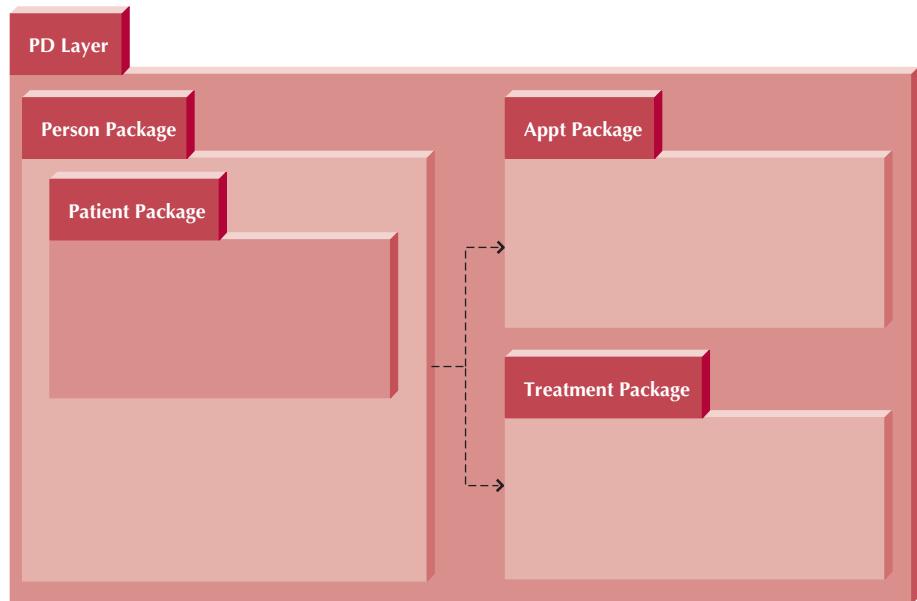


FIGURE 9-7
Overview Package
Diagram of the PD
Layer for the
Appointment System

Applying Concepts at CD Selections

In the previous chapters, the CD Selections Internet sales system has been described. In Chapter 5, the functional and nonfunctional requirements for the system were given in Figure 5-13. In Chapter 6, the functional model in the form of an activity diagram (see Figure 6-12) and the use case descriptions and diagram (see Figures 6-13 through 6-17). The structural model comprised of the CRC cards and class diagram was given in Chapter 7 (see Figures 7-9 through 7-11 and 7-13). Finally, in Chapter 8 behavioral models were developed for the Places Order use case and the Order class (see the sequence diagram in Figure 8-5, the communication diagram in Figure 8-10, the behavioral state machine in Figure 8-15). In this section, we demonstrate the creation of a package diagram for the CD Selections Internet sales system.

As just detailed, the first thing to do when identifying packages and creating package diagrams is to set the context. At this point in the development of the Internet sales system for CD Selections, Alec wants the development team only to concentrate on the Problem Domain Layer.

The second step, cluster classes together, is accomplished by reviewing the relationships among the different classes (see Figures 7-13, 8-5, and 8-10). Through this review process, we see that there are generalization, aggregation, various associations, and message sending relationships. Since we always want to keep classes in a generalization hierarchy together, we automatically cluster the Customer, Individual, and Organization classes together to form a partition. It is also preferred to keep classes together that participate in aggregation relationships. Based on aggregation relationships, we cluster the Mkt Info, Review, Artist Info, and Sample Clip classes together in a partition. Based on the association relationship and the message-sending pattern of activity (contained in the communication diagram in Figure 8-10) between the CD and Mkt Info classes, it seemed to the development team that these classes should be in the same partition. Furthermore, since the Vendor class is only related to the CD class (see the class diagram in Figure 7-13), the development team decided to place it in the same partition. Finally, the development team decided to place the Order and Order Item classes together and the Search Req and CD List classes together in their own partitions.

The third step was to model each of these partitions as packages. Figure 9-8 shows the classes being contained in their respective packages. Observe that the Credit-Card Center currently is not contained in any package.

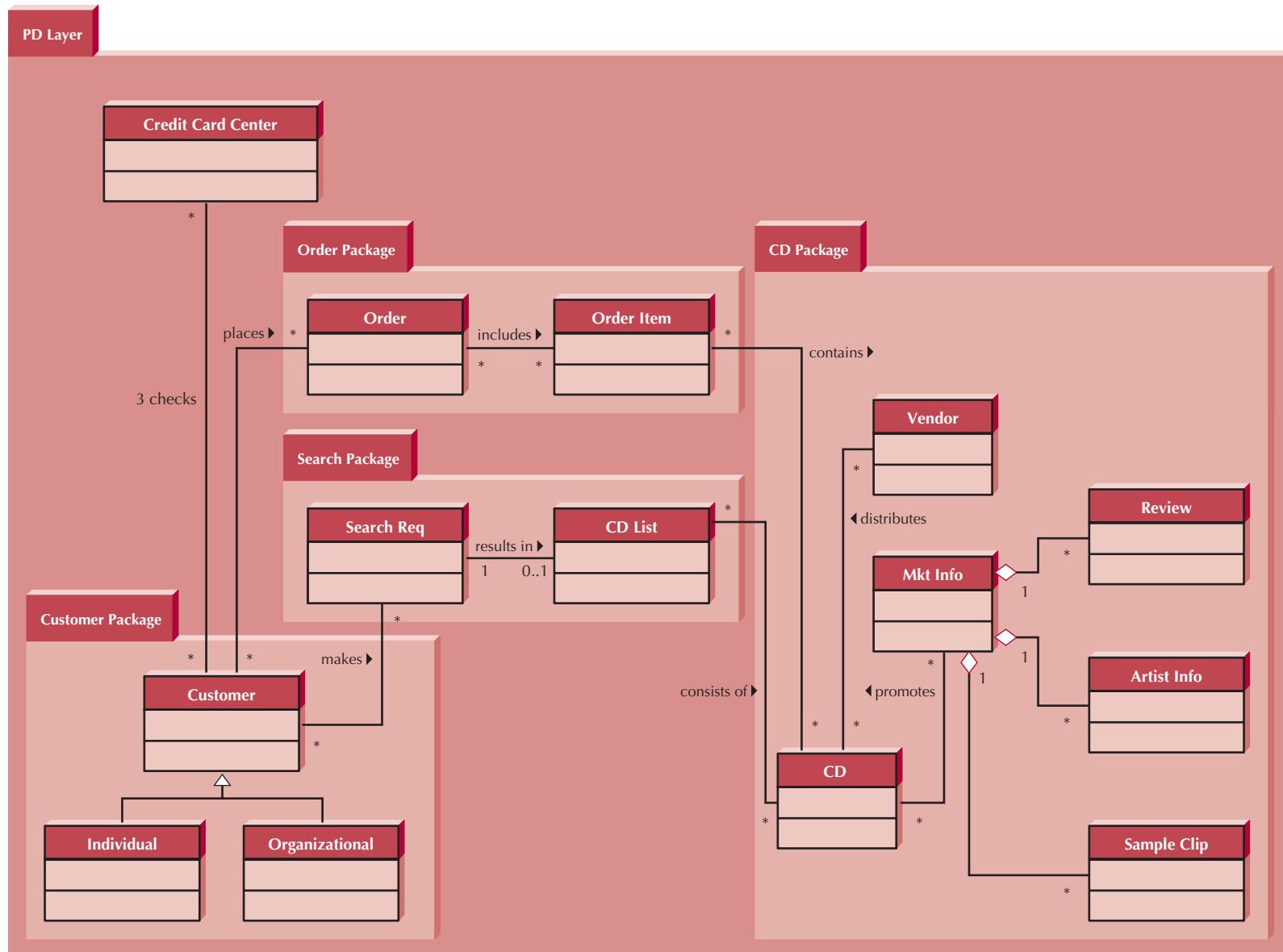


FIGURE 9-8 Package Diagram of the PD Layer of CD Selections Internet Sales System

Identifying dependency relationships among the packages is the fourth step. In this case, Alec was capable of quickly identifying four associations among the different packages: the Customer Package and the Order Package, the Customer Package and the Search Package, the Order Package and the CD Package, and the Search Package and the CD Package. He also identified an association between the Credit Card Clearance Center class and the Customer Package. Based on these associations, five dependency relationships were identified.

The fifth and final step is to place the dependency relationships on the package diagram. Again, to increase the understandability of the dependency relationships among the different packages, Alec decided to create a pure package diagram that only depicted the highest-level packages (and in this case the Credit Card Center class) and the dependency relationships (see Figure 9-9).

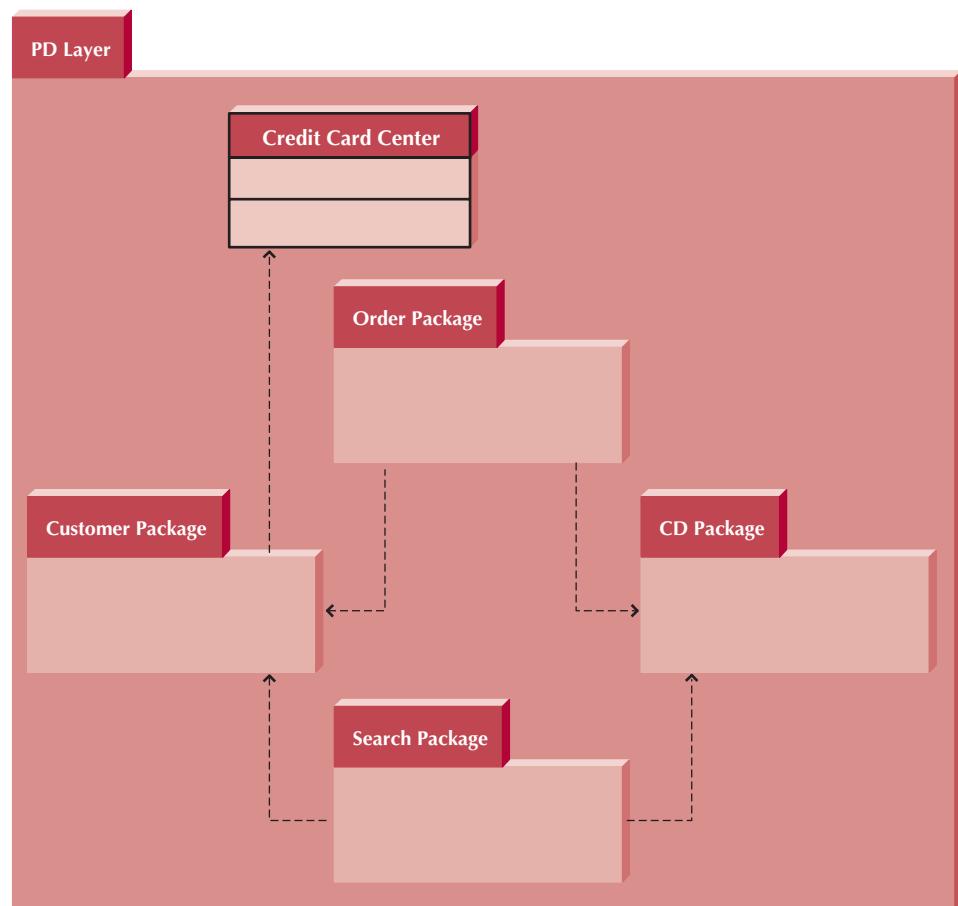


FIGURE 9-9
Overview Package
Diagram of the PD
Layer of CD Selections
Internet Sales System

YOUR TURN

9-2 Campus Housing

Based on the factoring of the evolving system in Your Turn 9-1, identify a set of partitions for the Problem Domain layer and model them in a package diagram.

DESIGN STRATEGIES

Until now, we have assumed that the system will be built and implemented by the project team; however, there are actually three ways to approach the creation of a new system: developing a custom application in-house, buying a packaged system and customizing it, and relying on an external vendor, developer, or service provider to build the system. Each of these choices has its strengths and weaknesses, and each is more appropriate in different scenarios. The following sections describe each design choice in turn, and then we present criteria that you can use to select one of the three approaches for your project.

Custom Development

Many project teams assume that *custom development*, or building a new system from scratch, is the best way to create a system. For one thing, teams have complete control over the way the system looks and functions. Let's consider the order taking process for CD Selections. If the company wanted a Web order-taking feature that links tightly with its existing distribution system, the project may involve a complex, highly specialized program. Or, CD Selections might have a technical environment in which all information systems are built using standard technology and interface designs so that they are consistent and easier to update and support. In both cases, it could be very effective to create a new system from scratch that meets these highly specialized requirements.

Custom development also allows developers to be flexible and creative in the way they solve business problems. CD Selections may envision the Web interface that takes customer orders as an important strategic enabler. The company may want to use the information from the system to better understand their customers who order over the Web, and it may want the flexibility to evolve the system to incorporate technology, such as data-mining software and geographic information systems to perform marketing research. A custom application would be easier to change to include components that take advantage of current technologies that can support such strategic efforts.

Building a system in-house also builds technical skills and functional knowledge within the company. As developers work with business users, their understanding of the business grows and they become better able to align IS with strategies and needs. These same developers climb the technology learning curve so that future projects applying similar technology become much less effortful.

Custom application development, however, also includes a dedicated effort that includes long hours and hard work. Many companies have a development staff that already is overcommitted to filling huge backlogs of systems requests, and they just do not have time for another project. Also, a variety of skills—technical, interpersonal, functional, project management, and modeling—all have to be in place for the project to move ahead smoothly. IS professionals, especially highly skilled individuals, are quite difficult to hire and retain.

The risks associated with building a system from the ground up can be quite high, and there is no guarantee that the project will succeed. Developers could be pulled away to work on other projects, technical obstacles could cause unexpected delays, and the business users could become impatient with a growing timeline.

Packaged Software

Many business needs are not unique, and because it makes little sense to “reinvent the wheel,” many organizations buy *packaged software* that has already been written, rather than developing their own custom solution. In fact, there are thousands of commercially available software programs that have already been written to serve a multitude of purposes. Think about your own need for a word processor—did you ever consider writing

CONCEPTS

IN ACTION

9-A Building a Custom System—with Some Help

I worked with a large financial institution in the southeast that suffered serious financial losses several years ago. A new CEO was brought in to change the strategy of the organization to being more “customer-focused.” The new direction was quite innovative, and it was determined that custom systems, including a data warehouse, would have to be built to support the new strategic efforts. The problem was that the company did not have the in-house skills for these kinds of custom projects.

The company now has one of the most successful data-warehouse implementations because of its willingness to use outside skills and its focus on project management. To supplement skills within the company, eight sets of external consultants, including hardware vendors, system integrators, and business strategists were hired to take part and transfer critical skills to internal employees. An in-house project manager coordinated the data-warehouse implementation full-time, and her primary goals were to set expectations clearly, define

responsibilities, and communicate the interdependencies that existed among the team members.

This company shows that successful custom development can be achieved, even when the company may not start off with the right skills in-house. But, this kind of project is not easy to pull off—it takes a talented project manager to keep the project moving along and to transition the skills to the right people over time.

—Barbara Wixom

Questions:

1. What are the risks in building a custom system without the right technical expertise? Why did the company select a project manager from within the organization? Would it have been better to hire an external professional project manager to coordinate the project?

your own word processing software? That would be very silly considering the number of good software packages available for a relatively inexpensive cost.

Similarly, most companies have needs that can be met quite well by packaged software such as payroll or accounts receivable. It can be much more efficient to buy programs that have already been created, tested, and proven, and a packaged system can be bought and installed in a relatively short period of time, when compared with a custom system. Plus, packaged systems incorporate the expertise and experience of the vendor who created the software.

Let's examine the Order Taking process once again. It turns out that there are programs available, called shopping cart programs, that allow a company to sell products on the Internet by keeping track of customers' selections, totaling them, and then e-mailing the order to a mailbox. You can easily install it on an existing Web page, and it allows you to take orders. Some shopping cart programs are even available over the Internet for free. Therefore, CD Selections could decide that acquiring a ready-made order taking application might be much more efficient than creating one from scratch.

Packaged software can range from reusable components (such as ActiveX and JavaBeans) to small single-function tools (like the shopping cart program) to huge, all-encompassing systems such as *enterprise resource planning (ERP)* applications that are installed to automate an entire business. Implementing ERP systems is a process in which large organizations spend millions of dollars installing packages by companies like SAP, PeopleSoft, Oracle, and Baan and then change their businesses accordingly. Installing ERP software is much more difficult than installing small application packages because benefits can be harder to realize and problems are much more serious.

One problem is that companies buying packaged systems must accept the functionality that is provided by the system, and rarely is there a perfect fit. If the packaged system is large in scope, its implementation could mean a substantial change in the way the company does business. Letting technology drive the business can be a dangerous way to go.

Most packaged applications allow for *customization*, or the manipulation of system parameters to change the way certain features work. For example, the package might have a way to accept information about your company or the company logo that would then appear on input screens. Or, an accounting software package could offer a choice of various ways to handle cash flow or inventory control so that it can support the accounting practices in different organizations. If the amount of customization is not enough, and the software package has a few features that don't quite work the way the company needs it to work, the project team can create workarounds.

A *workaround* is a custom-built add-on program that interfaces with the packaged application to handle special needs. It can be a nice way to create needed functionality that does not exist in the software package. But, workarounds should be a last resort for several reasons. First, workarounds are not supported by the vendor who supplied the packaged software, so when upgrades are made to the main system, they may make the workaround ineffective. Also, if problems arise, vendors have a tendency to blame the workaround as the culprit and refuse to provide support.

Although choosing a packaged software system is simpler than custom development, it too can benefit from following a formal methodology just as if you were building a custom application.

Systems integration refers to the process of building new systems by combining packaged software, existing legacy systems, and new software written to integrate these together. Many consulting firms specialize in systems integration, so it is not uncommon for companies to select the packaged software option and then outsource the integration of a variety of packages to a consulting firm. (Outsourcing is discussed in the next section.)

The key challenge in systems integration is finding ways to integrate the data produced by the different packages and legacy systems. Integration often hinges around taking data produced by one package or system and reformatting it for use in another package/system. The project team starts by examining the data produced by and needed by the different packages/systems and identifying the transformations that must occur to move the data from one to the other. In many cases, this involves "fooling" the different packages/systems into thinking that the data was produced by an existing program module that the package/system expects to produce the data rather than the new package/system that is being integrated.

For example, CD Selections needs to integrate the new Internet sales system with existing legacy systems, such as the distribution system. The distribution system was written to support a different application: the distribution of CDs to retail stores, and it currently exchanges data with the system that supports the CD Selections retail stores. The Internet sales system will need to produce data in the same format as this existing system so the distribution system will think the data is from the same system as always. Conversely, the project team may need to revise the existing distribution system, so it can accept data from the Internet sales system in a new format. A third approach is through the use of an *object wrapper*.¹¹ An object wrapper essentially is an object that "wraps around" a legacy system, enabling an object-oriented system to send messages to the legacy system. Effectively, object wrappers create an application program interface (API) to the legacy system. The creation of an object wrapper allows the protection of the corporation's investment in the legacy system.

Outsourcing

The design choice that requires the least amount of in-house resources is *outsourcing*—hiring an external vendor, developer, or service provider to create the system. Outsourcing has

¹¹ Ian Graham, *Object-Oriented Methods: Principles & Practice*, 3rd Ed. (Reading, MA: Addison-Wesley, 2001).

become quite popular in recent years. Some estimate that as many as 50 percent of companies with IT budgets of more than \$5 million are currently outsourcing or evaluating the approach. Although these figures include outsourcing for all kinds of systems functions, this section focuses on outsourcing a single development project.

There can be great benefit to having someone else develop your system. They may be more experienced in the technology or have more resources, like experienced programmers. Many companies embark upon outsourcing deals to reduce costs, while others see it as an opportunity to add value to the business. For example, instead of creating a program that handles the Order Taking Process or buying a preexisting package, CD Selections may decide to let a Web service provider provide commercial services for them.

For whatever reason, outsourcing can be a good alternative for a new system. However, it does not come without costs. If you decide to leave the creation of a new system in the hands of someone else, you could compromise confidential information or lose control over future development. In-house professionals are not benefiting from the skills that could be learned from the project, and instead the expertise is transferred to the outside organization. Ultimately, important skills can walk right out the door at the end of the contract.

Most risks can be addressed if you decide to outsource, but two are particularly important. First, assess the requirements for the project thoroughly—you should never outsource what you don't understand. If you have conducted rigorous planning and analysis, then you should be well aware of your needs. Second, carefully choose a vendor, developer, or service with a proven track record with the type of system and technology that your system needs.

Three primary types of contracts can be drawn to control the outsourcing *contract*. A *time and arrangements* deal is very flexible because you agree to pay for whatever time and expenses are needed to get the job done. Of course, this agreement could result in a large bill that exceeds initial estimates. This works best when you and the outsourcer are unclear about what it is going to take to finish the job.

You will pay no more than expected with a *fixed-price contract* because if the outsourcer exceeds the agreed-upon price, they will have to absorb the costs. Outsourcers are much more careful about defining requirements clearly up front, and there is little flexibility for change.

The type of contract gaining in popularity is the *value-added contract* whereby the outsourcer reaps some percentage of the completed system's benefits. You have very little risk in this case, but expect to share the wealth once the system is in place.

Creating fair contracts is an art because you need to carefully balance flexibility with clearly defined terms. Often needs change over time. As such, you don't want the contract to be so specific and rigid that alterations can't be made. Think about how quickly technology like the World Wide Web changes. It is difficult to foresee how a project may evolve over a long period of time. Short-term contracts help leave room for reassessment if needs change or if relationships are not working out the way both parties expected. In all cases, the relationship with the outsourcer should be viewed as a partnership where both parties benefit and communicate openly.

Managing the outsourcing relationship is a full-time job. Thus, someone needs to be assigned full-time to manage the outsourcer, and the level of that person should be appropriate for the size of the job (a multimillion dollar outsourcing engagement should be handled by a high level executive). Throughout the relationship, progress should be tracked and measured against predetermined goals. If you do embark upon an outsourcing design strategy, be sure to get more information. Many books have been written that provide much more detailed information on the topic.¹² Figure 9-10 summarizes some guidelines for outsourcing.

¹² For more information on outsourcing, we recommend Lacity, M. and Hirschheim, R., *Information Systems Outsourcing: Myths, Metaphors, and Realities* (New York, NY: Wiley, 1993) and Willcocks, L. and Fitzgerald, G., *A Business Guide to Outsourcing Information Technology* (London: Business Intelligence, 1994).

CONCEPTS

9-B EDS Value-Added Contract

IN ACTION

Value-added contracts can be quite rare—and very dramatic. They exist when a vendor is paid a percentage of revenue generated by the new system, which reduces the up-front fee, sometimes to zero. The City of Chicago and EDS (a large consulting and systems integration firm) agreed to reengineer the process by which the city collects the fines on 3.6 million parking tickets per year, and thus signed a landmark deal of this type three years ago. At the time, because of clogged courts and administrative problems, the city collected on only about 25 percent of all tickets issued. It had a \$60 million backlog of uncollected tickets.

Dallas-based EDS invested an estimated \$25 million in consulting and new systems in exchange for the right to up to 26 percent of the uncollected fines, a base

processing fee for new tickets, and software rights. To date, EDS has taken in well over \$50 million on the deal, analysts say. The deal has come under some fire from various quarters as an example of an organization giving away too much in a risk/reward-sharing deal. City officials, however, counter that the city has pulled in about \$45 million in previously uncollected fines and has improved its collection rate to 65 percent with little up-front investment.

Source: *Datamation*, February 1, 1995.

Question:

1. Do you think the City of Chicago got a good deal from this arrangement? Why or why not?

Selecting a Design Strategy

Each of the design strategies just discussed has its strengths and weaknesses, and no one strategy is inherently better than the others. Thus, it is important to understand the strengths and weaknesses of each strategy and when to use each. Figure 9-11 presents a summary of the characteristics of each strategy.

Business Need If the business need for the system is common, and technical solutions already exist in the marketplace that can meet the business need of the system, it makes little sense to build a custom application. Packaged systems are good alternatives for common business needs. A custom alternative will need to be explored when the business need is unique or has special requirements. Usually if the business need is not critical to the company, then outsourcing is the best choice—someone outside of the organization can be responsible for the application development.

In-house Experience If in-house experience exists for all of the functional and technical needs of the system, it will be easier to build a custom application than if these skills do not exist. A packaged system may be a better alternative for companies that do not have the technical skills to build the desired system. For example, a project team that does not have Web commerce technology skills may want to acquire a Web commerce package that can

Outsourcing Guidelines

- Keep the lines of communication open between you and your outsourcer.
- Define and stabilize requirements before signing a contact.
- View the outsourcing relationship as a partnership.
- Select the vendor, developer or service provider carefully.
- Assign a person to managing the relationship.
- Don't outsource what you don't understand.
- Emphasize flexible requirements, long-term relationships and short-term contracts.

FIGURE 9-10
Outsourcing Guidelines

	Use Custom Development When...	Use a Packaged System When...	Use Outsourcing When...
Business Need	The business need is unique.	The business need is common.	The business need is not core to the business.
In-house Experience	In-house functional and technical experience exists.	In-house functional experience exists.	In-house functional or technical experience does not exist.
Project Skills	There is a desire to build in-house skills.	The skills are not strategic.	The decision to outsource is a strategic decision.
Project Management	The project has a highly skilled project manager and a proven methodology.	The project has a project manager who can coordinate vendor's efforts.	The project has a highly skilled project manager at the level of the organization that matches the scope of the outsourcing deal.
Timeframe	The timeframe is flexible.	The timeframe is short.	The timeframe is short or flexible.

FIGURE 9-11 Selecting a Design Strategy

be installed without many changes. Outsourcing is a good way to bring in outside experience that is missing in-house so that skilled people are in charge of building the system.

Project Skills The skills that are applied during projects are either technical (e.g., Java, SQL) or functional (e.g., electronic commerce), and different design alternatives are more viable, depending on how important the skills are to the company's strategy. For example, if certain functional and technical expertise that relate to Internet sales applications and Web commerce application development are important to the organization because it expects the Internet to play an important role in its sales over time, then it makes sense for the company to develop Web commerce applications in-house, using company employees so that the skills can be developed and improved. On the other hand, some skills like network security may be either beyond the technical expertise of employees or not of interest to the company's strategists—it is just an operational issue that needs to be addressed. In this case, packaged systems or outsourcing should be considered so internal employees can focus on other business-critical applications and skills.

Project Management Custom applications require excellent project management and a proven methodology. There are so many things like funding obstacles, staffing hold-ups, and overly demanding business users that can push a project off-track. Therefore, the project team should choose to develop a custom application only if they are certain that the underlying coordination and control mechanisms will be in place. Packaged and outsourcing alternatives also need to be managed; however, they are more shielded from internal obstacles because the external parties have their own objectives and priorities (e.g., it may be easier for an outside contractor to say "no" to a user than a person within the company). The latter alternatives typically have their own methodologies, which can benefit companies that do not have an appropriate methodology to use.

Timeframe When time is a factor, the project team should probably start looking for a system that is already built and tested. In this way, the company will have a good idea of how long the package will take to put in place and what the final result will contain. The timeframe for custom applications is hard to pin down, especially when you consider how many projects end up missing important deadlines. If you must choose the custom development alternative, and the timeframe is very short, consider using techniques like timeboxing to manage this problem. The time to produce a system using outsourcing really

depends on the system and the outsourcer's resources. If a service provider has services in place that can be used to support the company's needs, then a business need could be implemented quickly. Otherwise, an outsourcing solution could take as long as a custom development initiative.

DEVELOPING THE ACTUAL DESIGN

Once the project team has a good understanding of how well each design strategy fits with the project's needs, they must begin to understand exactly *how* to implement these strategies. For example, what tools and technology would be used if a custom alternative were selected? What vendors make packaged systems that address the project needs? What service providers would be able to build this system if the application were outsourced? This information can be obtained from people working in the IS department and from recommendations by business users. Or, the project team can contact other companies with similar needs and investigate the types of systems that they have put in place. Vendors and consultants usually are willing to provide information about various tools and solutions in the form of brochures, product demonstrations, and information seminars. However, be sure to validate the information you receive from vendors and consultants. After all, they are trying to make a sale. Therefore, they may "stretch" the capabilities of their tool by only focusing on the positive aspects of the tool while omitting the tool's drawbacks.

It is likely that the project team will identify several ways that the system could be constructed after weighing the specific design options. For example, the project team may have found three vendors that make packaged systems that potentially could meet the project's needs. Or, the team may be debating over whether to develop a system using Java as a development tool and the database management system from Oracle; or to outsource the development effort to a consulting firm like Accenture or American Management Systems. Each alternative will have pros and cons associated with it that need to be considered, and only one solution can be selected in the end.

Alternative Matrix

An *alternative matrix* can be used to organize the pros and cons of the design alternatives so that the best solution will be chosen in the end. This matrix is created using the same steps as the feasibility analysis, which was presented in Chapter 3. The only difference is that the alternative matrix combines several feasibility analyses into one matrix so that the alternatives can be easily compared. The alternative matrix is a grid that contains the technical, budget, and organizational feasibilities for each system candidate, pros and cons associated with adopting each solution, and other information that is helpful when making comparisons. Sometimes weights are provided for different parts of the matrix to show when some criteria are more important to the final decision.

YOUR TURN

9-3 Choose a Design Strategy

Suppose that your university was interested in creating a new course registration system that can support Web-based registration. What should the university consider

when determining whether to invest in a custom, packaged, or outsourced system solution?

To create the alternative matrix, draw a grid with the alternatives across the top and different criteria (e.g., feasibilities, pros, cons, and other miscellaneous criteria) along the side. Next, fill in the grid with detailed descriptions about each alternative. This becomes a useful document for discussion because it clearly presents the alternatives being reviewed and comparable characteristics for each one.

Suppose that your company is thinking about implementing a packaged financial system like Oracle Financials or Microsoft's Great Plains, but there is not enough expertise in-house to be able to create a thorough alternative matrix. This situation is quite common—often the alternatives for a project are unfamiliar to the project team, so outside expertise is needed to provide information about the alternatives' criteria.

One helpful tool is the *request for proposals (RFP)*. An RFP is a document that solicits proposals to provide the alternative solutions from a vendor, developer, or service provider. Basically, the RFP explains the system that you are trying to build and the criteria that you will use to select a system. Vendors then respond by describing what it would mean for them to be a part of the solution. They communicate the time, cost, and exactly how their product or services will address the needs of the project.

There is no formal way to write an RFP, but it should include basic information like the description of the desired system, any special technical needs or circumstances, evaluation criteria, instructions for how to respond, and the desired schedule. An RFP can be a very large document (i.e., hundreds of pages) because companies try to include as much detail as possible about their needs so that the respondent can be just as detailed in the solution that would be provided. Thus, RFPs typically are used for large projects rather than small ones because they take a lot of time to create, and even more time and effort for vendors, developers, and service providers to develop high quality responses—only a project with a fairly large price tag would be worth the time and cost to develop a response for the RFP.

A less effort-intensive tool is a *request for information (RFI)* that includes the same format as the RFP. The RFI is shorter and contains less detailed information about a company's needs, and it requires general information from respondents that communicates the basic services that they can provide.

The final step, of course, is to decide which solution to design and implement. The decision should be made by a combination of business users and technical professionals after the issues involved with the different alternatives are well understood. Once the decision is finalized, the design phase can continue as needed, based on the selected alternative.

Applying the Concepts at CD Selections

Alec had three different approaches that he could take with the new system: he could develop the entire system using development resources from CD Selections, he could buy

YOUR TURN

9-4 Alternative Matrix

Pretend that you have been assigned the task of selecting a CASE tool for your class to use for a semester project. Using the Web or other references resources, select three CASE tools (e.g., ArgoUML, Poseidon, Rational Rose, or

Visual Paradigm). Create an alternative matrix that can be used to compare the three software products in a way in which a selection decision can be made.

a commercial Internet sales packaged software program (or a set of different packages and integrate them), or he could hire a consulting firm or service provider to create the system. Immediately, Alec ruled out the third option. Building Internet applications, especially sales systems, was important to the CD Selections' business strategy. By outsourcing the Internet sales system, CD Selections would not develop Internet application development skills and business skills within the organization.

Instead, Alec decided that a custom development project using the company's standard Web development tools would be the best choice for CD Selections. In this way, the company would be developing critical technical and business skills in-house, and the project team would be able to have a high level of flexibility and control over the final product. Also, Alec wanted the new Internet sales system to directly interface with the existing distribution system, and there was a chance that a packaged solution would not be able to integrate as well into the CD Selections environment.

There was one part of the project that potentially could be handled using packaged software: the shopping cart portion of the application. Alec realized that a multitude of programs have been written and are available (at low prices) to handle a customer's order transaction over the Web. These programs allow customers to select items for an order form, input credit card and billing information, and finalize the order transaction. Alec believed that the project team should at least consider some of these packaged alternatives so that less time had to be spent writing a program that handled basic Web tasks, and more time could be devoted to innovative marketing ideas and custom interfaces with the distribution system.

To help better understand some of the shopping cart programs that were available in the market and how their adoption could benefit the project, Alec created an alternative matrix that compared three different shopping cart programs to one another (see Figure 9-12). Although all three alternatives had positive points, Alec saw Alternative B (WebShop) as the best alternative for handling the shopping cart functionality for the new Internet sales system. WebShop was written in JAVA, the tool that CD Selections selected as its standard Web development language; the expense was reasonable, with no hidden or recurring costs; and there was a person in-house who had some positive experience with the program. Alec made a note to look into acquiring WebShop as the shopping cart program for the Internet sales system.

SUMMARY

The design phase contains many steps that guide the project team through planning out exactly how the system needs to be constructed. The requirements that were identified and the models that were created in the analysis phase serve as the primary inputs for the design activities. In object-oriented design, the primary activity is to evolve the analysis models into design models by optimizing the problem domain information already contained in the analysis models and adding system environment details to them.

Evolving the Analysis Models into Design Models

When evolving the analysis models into design models, you should first carefully review the analysis models: activity diagrams, use case descriptions, use case diagrams, CRC cards, class and object diagrams, sequence diagrams, communication diagrams, and behavioral state machines. During this review, factoring, refinement, and abstraction processes can be used to polish the current models. During this polishing, it is possible that the analysis models may become overly complex. If this occurs, then the models

	Alternative 1: Shop-With-Me	Alternative 2: WebShop	Alternative 3: Shop-N-Go
Technical Feasibility	<ul style="list-style-type: none"> Developed using C: very little C experience in-house Orders sent to company using email files 	<ul style="list-style-type: none"> Developed using C and JAVA: would like to develop in-house JAVA skills Flexible export features for passing order information to other systems 	<ul style="list-style-type: none"> Developed using JAVA: would like to develop in-house JAVA skills Orders saved to a number of file formats
Economic Feasibility	<ul style="list-style-type: none"> \$150 initial charge 	<ul style="list-style-type: none"> \$700 up front charge, no yearly fees 	<ul style="list-style-type: none"> \$200/year
Organizational Feasibility	<ul style="list-style-type: none"> Program used by other retail music companies 	<ul style="list-style-type: none"> Program used by other retail music companies 	<ul style="list-style-type: none"> Brand new application: few companies have experience with Shop-N-Go to date
Other Benefits	<ul style="list-style-type: none"> Very simple to use 	<ul style="list-style-type: none"> Tom in IS support has had limited, but positive experience with this program Easy to customize 	
Other Limitations			<ul style="list-style-type: none"> The interface is not easily customized

FIGURE 9-12 Alternative Matrix for Shopping Cart Program

should be partitioned based on the interactivity (message sending) and relationships (generalization, aggregation, and association) shared among the classes. The more a class has in common with another class (i.e., the more relationships shared), the more likely they belong in the same partition.

The second thing to do to evolve the analysis model is to add the system environment (physical architecture, user interface, and data access and management) information to the problem domain information already contained in the model. To accomplish this and to control the complexity of the models, layers are used. A layer represents an element of the software architecture of the system. We recommend five different layers to be used: foundation, physical architecture, human computer interaction, data access and management, and problem domain. Each layer only supports certain types of classes (e.g., data access manipulation classes would only be allowed on the data management layer).

Packages and Package Diagrams

A package is a general UML construct used to represent collaborations, partitions, and layers. Its primary purpose is to support the logical grouping of other UML constructs together (e.g., use cases and classes), by the developer and user to simplify and increase the understandability of a UML diagram. There are instances in which a diagram that contains only packages is useful. A package diagram contains packages and dependency relationships. A dependency relationship represents the possibility of a modification dependency existing between two packages (i.e., changes in one package could cause changes in the dependent package).

Identifying packages and creating a package diagram is accomplished using a five-step process. The five steps can be summed up as setting the context, clustering similar classes, placing the clustered classes into a package, identifying dependency relationships among the packages, and placing the dependency relationship on the package diagram.

Design Strategies

During the design phase, the project team also needs to consider three approaches to creating the new system, including developing a custom application in-house, buying a packaged system and customizing it, and relying on an external vendor, developer or system provider to build and/or support the system.

Custom development allows developers to be flexible and creative in the way they solve business problems, and it builds technical and functional knowledge within the organization. But, many companies have a development staff that is already overcommitted to filling huge backlogs of systems requests, and they just don't have time to devote to a project where a system is built from scratch. It can be much more efficient to buy programs that have been created, tested, and proven, and a packaged system can be bought and installed in a relatively short period of time, when compared with a custom solution. Workarounds can be used to meet the needs that are not addressed by the packaged application.

The third design strategy is to outsource the project and pay an external vendor, developer or service provider to create the system. It can be a good alternative for how to approach the new system; however, it does not come without costs. However, if a company decides to leave the creation of a new system in the hands of someone else, the organization could compromise confidential information or lose control over future development.

Each of the design strategies discussed above has its strengths and weaknesses, and no one strategy is inherently better than the others. Thus, it is important to consider such issues as the uniqueness of business need for the system, the amount of in-house experience that is available to build the system, and the importance of the project skills to the company. Also, the existence of good project management and the amount of time available to develop the application play a role in the selection process.

Developing the Actual Design

Ultimately, the decision must be made regarding the specific type of system that needs to be designed. An alternative matrix can help make this decision by presenting feasibility information for several candidate solutions in a way in which they can be compared easily. Both the Request for Proposal and Request for Information are two ways to gather accurate information regarding the alternatives.

KEY TERMS

A-Kind-Of	Enterprise resource systems (ERP)	Package
Abstract classes	Factoring	Package diagram
Abstraction	Fixed-price contract	Packaged software
Aggregation	Foundation layer	Partition
Alternative matrix	Generalization	Physical architecture layer
Class	Has-Parts	Problem domain layer
Client	Human computer interaction layer	Refinement
Collaboration	Layer	Request for information
Concrete classes	Message	Request for proposals
Contract	Method	Server
Controller	Middleware	Smalltalk
Custom development	Model	Systems integration
Customization	Model-View-Controller (MVC)	Time and arrangements contract
Data management layer	Module	Value-added contract
Dependency relationship	Object wrapper	View
Design phase	Outsourcing	Workaround

QUESTIONS

1. What is the primary difference between an analysis model and a design model?
2. What does factoring mean? How is it related to abstraction and refinement?
3. What is a partition? How does a partition relate to a collaboration?
4. What is a layer? Name the different layers.
5. What is a package? How are packages related to partitions and layers?
6. What is a dependency relationship? How do you identify them?
7. What are the five steps for identifying packages and creating package diagrams?
8. What situations are most appropriate for a custom development design strategy?
9. What are some problems with using a packaged software approach to building a new system? How can these problems be addressed?
10. Why do companies invest in ERP systems?
11. What are the pros and cons of using a workaround?
12. When is outsourcing considered a good design strategy? When is it not appropriate?
13. What are the differences between the time and arrangements, fixed-price, and value-added contracts for outsourcing?
14. How are the alternative matrix and feasibility analysis related?
15. What is an RFP? How is this different from an RFI?

EXERCISES

- A. Based on the functional models you created for exercises E, F, and G in Chapter 6 and the structural model you created in Exercise K in Chapter 7, draw a package diagram for the problem domain layer for the dentist office system.
- B. Based on the functional models you created for exercises J and K in Chapter 6 and the structural model you created in Exercise M in Chapter 7, draw a package diagram for the problem domain layer for the real estate system.
- C. Based on the functional models you created for exercises L and M in Chapter 6, the structural model you created in Exercise N in Chapter 7, and the behavioral models you created in exercises A and B in Chapter 8, draw a package diagram for the problem domain layer for the video store system.
- D. Based on the functional models you created for exercises N and O in Chapter 6, the structural model you created in Exercise O in Chapter 7, and the behavioral models you created in exercises C and D in Chapter 8, draw a package diagram for the problem domain layer for the health club membership system.
- E. Based on the functional models you created for exercises P and Q in Chapter 6, the structural model you created in Exercise P in Chapter 7, and the behavioral models you created in exercises H in Chapter 8, draw a package diagram for the problem domain layer for the catering system.
- F. Based on the functional models you created for exercises R and S in Chapter 6, the structural model you created in Exercise Q in Chapter 7, and the behavioral models you created in exercises I in Chapter 8, draw a package diagram for the problem domain layer for the Of-the-Month Club.
- G. Based on the functional models you created for exercises T and U in Chapter 6, the structural model you created in Exercise R in Chapter 7, and the behavioral models you created in exercises J in Chapter 8, draw a package diagram for the problem domain layer for the university library system.
- H. Which design strategy would you recommend for the construction of this system in:
 - a. Exercise A. Why?
 - b. Exercise B. Why?
 - c. Exercise C. Why?
 - d. Exercise D. Why?
 - e. Exercise E. Why?
 - f. Exercise F. Why?
 - g. Exercise G. Why?
- I. Pretend that you are leading a project that will implement a new course enrollment system for your university. You are thinking about either using a packaged course enrollment application or outsourcing the job to an external consultant. Create an outline for a Request for Proposal to which interested vendors and consultants could respond.
- J. Pretend that you and your friends are starting a small business painting houses in the summertime. You need to buy a software package that handles the financial transactions of the business. Create an alternative matrix that compares three packaged systems (e.g., Quicken, MS Money, Quickbooks). Which alternative appears to be the best choice?

MINICASES

1. Susan, president of MOTO, Inc., a human resources management firm, is reflecting on the client management software system her organization purchased four years ago. At that time, the firm had just gone through a major growth spurt, and the mixture of automated and manual procedures that had been used to manage client accounts became unwieldy. Susan and Nancy, her IS department head, researched and selected the package that is currently used. Susan had heard about the software at a professional conference she attended, and at least initially, it worked fairly well for the firm. Some of their procedures had to change to fit the package, but they expected that and were prepared for it.

Since that time, MOTO, Inc. has continued to grow, not only through an expansion of the client base, but also through the acquisition of several smaller employment-related businesses. MOTO, Inc. is a much different business than it was four years ago. Along with expanding to offer more diversified human resource management services, the firm's support staff has also expanded. Susan and Nancy are particularly proud of the IS department they have built up over the years. Using strong ties with a local university, an attractive compensation package, and a good working environment, the IS department is well-staffed with competent, innovative people, plus a steady stream of college interns that keeps the department fresh and lively. One of the IS teams pioneered the use of the Internet to offer MOTO's services to a whole new market segment, an experiment that has proven very successful.

It seems clear that a major change is needed in the client management software, and Susan has already begun to plan financially to undertake such a project. This software is a central part of MOTO's operations, and Susan wants to be sure that a quality system is obtained this time. She knows that the vendor of their current system has made some revisions and additions to its product line. There are also a number of other software vendors who offer products that may be suitable. Some of these vendors did not exist when the purchase was made four years ago. Susan is also

considering Nancy's suggestion that the IS department develop a custom software application.

- a. Outline the issues that Susan should consider that would support the development of a custom software application in-house.
 - b. Outline the issues that Susan should consider which would support the purchase of a software package.
 - c. Within the context of a systems development project, when should the decision of "make-versus-buy" be made? How should Susan proceed? Explain your answer.
2. Refer to Minicase 1 in Chapter 7. After completing all of the analysis models (both the As-Is and To-Be models) for West Star Marinas, the Director of Operations finally understood why it was important to understand the As-Is system before delving into the development of the To-Be system. However, you now tell him that the To-Be models are only the problem domain portion of the design. To say the least, he is now very confused. After explaining to him the advantages of using a layered approach to developing the system, he informs you that "I don't care about reusability or maintenance. I only want the system to be implemented as soon as possible. You IS types are always trying to pull a fast one on the users. Just get the system completed."
- What is your response to the Director of Operations? Do you jump into implementation as he seems to want? What do you do next?
3. Refer to the analysis models that you created for Professional and Scientific Staff Management (PSSM) for Minicase 2 in Chapter 6 and for Minicase 1 in Chapter 8.
 - a. Add packages to your use-case diagram to simplify it.
 - b. Use the communication diagram to identify logical partitions in your class diagram. Add packages to the diagram to represent the partitions.
4. Refer to the analysis models that you created for Holiday Travel Vehicles for Minicase 2 in Chapter 7 and for Minicase 2 in Chapter 8.
 - a. Add packages to your use-case diagram to simplify it.
 - b. Use the communication diagram to identify logical partitions in your class diagram. Add packages to the diagram to represent the partitions.

CHAPTER 10

CLASS AND METHOD DESIGN

The most important step of the design phase is designing the individual classes and methods. Object-oriented systems can be quite complex, so analysts need to create instructions and guidelines for programmers that clearly describe what the system must do. This chapter presents a set of criteria, activities, and techniques used to design classes and methods. Together they are used to ensure the object-oriented design communicates how the system needs to be coded.

OBJECTIVES

- Become familiar with coupling, cohesion, and connascence.
- Be able to specify, restructure, and optimize object designs.
- Be able to identify the reuse of predefined classes, libraries, frameworks, and components.
- Be able to specify constraints and contracts.
- Be able to create a method specification.

CHAPTER OUTLINE

Introduction	Restructuring the Design
Revisiting the Basic Characteristics of Object-Orientation	Optimizing the Design
Classes, Objects, Methods, and Messages	Mapping Problem Domain Classes to Implementation Languages
Encapsulation and Information Hiding	Constraints and Contracts
Polymorphism and Dynamic Binding	Types of Constraints
Inheritance	Elements of a Contract
Design Criteria	Method Specification
Coupling	General Information
Cohesion	Events
Connascence	Message Passing
Object Design Activities	Algorithm Specification
Additional Specification	Applying the Concepts at CD Selections
Identifying Opportunities for Reuse	Summary

INTRODUCTION

Class and Method design is where all of the work actually gets done during the design phase. No matter which layer you are focusing on, the classes, which will be used to create the system objects, must be designed. Some people believe that with reusable class libraries and off-the-shelf components, this type of low-level, or detailed, design is a waste of time and that we should jump immediately into the “real” work: coding the system. However, if

the past shows us anything, it shows that low-level or detailed design is critical despite the use of libraries and components. Detailed design is still very important for two reasons. First, even preexisting classes and components need to be understood, organized, and pieced together. Second, it is still common for the project team to have to write some code (if not all) and produce original classes that support the application logic of the system.

Jumping right into coding will guarantee results that can be disastrous. For example, even though the use of layers can simplify the individual classes, they can increase the complexity of the interactions between them. As such, if the classes are not designed carefully, the resulting system can be very inefficient. Or worse, the instances of the classes (i.e., the objects) will not be capable of communicating with each other, which, of course, will cause the system not to work properly.

Furthermore, in an object-oriented system, changes can take place at different levels of abstraction. These levels include Variable, Method, Class/Object, Cluster,¹ Library, and/or Application/System levels (see Figure 10-1). The changes that take place at one level can impact other levels (e.g., changes to a class can affect the cluster level, which can affect both the system level and the library level, which in turn can cause changes back down at the class level). Finally, changes can be occurring at different levels at the same time.

The good news is that the detailed design of the individual classes and methods is fairly straightforward and the interactions among the objects on the problem domain layer have been designed, in some detail, in the analysis phase (see Chapters 6 through 8). As far as the other layers go (system architecture, human computer interaction, and data management), they will be highly dependent on the problem domain layer. Therefore, if we get the problem domain classes designed correctly, the design of the classes on the other layers will fall into place, relatively speaking.

That being said, it has been our experience that many project teams are much too quick at jumping into writing code for the classes without first designing them. Some of

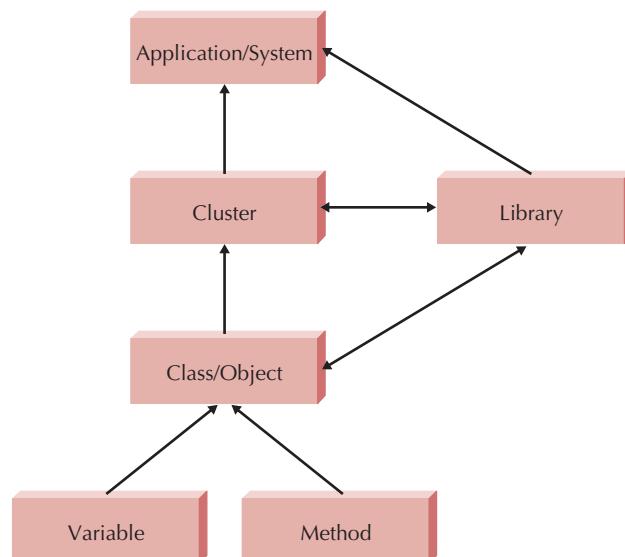


FIGURE 10-1
Levels of Abstraction
in Object-Oriented
Systems

Source: Adapted from David P. Tegarden, Steven D. Sheetz, and David E. Monarchi, "A Software Complexity Model of Object-Oriented Systems," *Decision Support Systems* 13 (March 1995): 241–262.

¹ A cluster is a group of collaborating objects. Other names for a cluster include package, partition, pattern, subject, and subsystem.

this has been caused by the fact that object-oriented systems analysis and design has evolved from object-oriented programming. Until recently there has been a general lack of accepted guidelines on how to design and develop effective object-oriented systems. However, with the acceptance of UML as a standard object notation, standardized approaches based on work of many object methodologists have begun to emerge.²

In this chapter, we begin by revisiting the principles of object-orientation. Next, we present a set of useful design criteria and activities that are applicable across any layer for class and method design. Finally, we present a set of techniques that are useful to design methods: contracts and method specifications.

REVISITING THE BASIC CHARACTERISTICS OF OBJECT-ORIENTATION

As pointed out in Chapter 2, object-oriented systems can be traced back to the Simula and Smalltalk programming languages. However, until the increase in processor power and the decrease in processor cost that occurred in the 1980s, object-oriented approaches were not practical. In this section, we review the basic characteristics of object-orientation.

Classes, Objects, Methods, and Messages

The basic building block of the system is the *object*. Objects are *instances* of *classes* that we use as templates to define objects. A class defines both the data and processes that each object contains. Each object has *attributes* that describe data about the object. Objects have *state*, which is defined by the value of its attributes and its relationships with other objects at a particular point in time. And, each object has *methods*, which specify what processes the object can perform. In order to get an object to perform a method (e.g., to delete itself), a *message* is sent to the object. A message is essentially a function or procedure call from one object to another object.

Encapsulation and Information Hiding

Encapsulation is the mechanism that combines the processes and data into a single object. *Information hiding* suggests only the information required to use an object be available outside the object. Exactly how the object stores data or performs methods is not relevant, as long as the object functions correctly. All that is required to use an object is the set of methods and the messages needed to be sent to trigger them. The only communication between objects should be through an object's methods. The fact that we can use an object by sending a message that calls methods is the key to reusability since it shields the internal workings of the object from changes in the outside system, and it keeps the system from being affected when changes are made to an object.

Polymorphism and Dynamic Binding

Polymorphism means having the ability to take several forms. By supporting polymorphism, object-oriented systems can send the same message to a set of objects, which can be interpreted differently by different classes of objects. And, based on encapsulation and information hiding, an object does not have to be concerned with *how* something is done when using other objects. It simply sends a message to an object and that object determines how to interpret the message. This is accomplished through the use of dynamic binding.

² For example, OPEN (I. Graham, B. Henderson-Seller, and H. Yanoussi, *The Open Process Specification* [Reading, MA: Addison-Wesley, 1997]) and RUP (P. Kruchten, *The Rational Unified Process: An Introduction*, 2nd Ed. [Reading, MA: Addison-Wesley, 2000]).

Dynamic binding refers to the ability of object-oriented systems to defer the data typing of objects to run time. For example, imagine that you have an array of type employee that contains instances of hourly employees and salaried employees. Both of these types of employees implement a “compute pay” method. An object can send the message to each instance contained in the array to compute the pay for that individual instance. Depending on whether the instance is an hourly employee or a salaried employee, a different method would be executed. The specific method is chosen at run time. With this ability, individual classes are easier to understand. However, the specific level of support for polymorphism and dynamic binding is language specific. Most object-oriented programming languages support dynamic binding of methods, and some support dynamic binding of attributes. As such, it is important to know what object-oriented programming language is going to be used.

But polymorphism can be a double-edged sword. Through the use of dynamic binding, there is no way to know before run time which specific object will be asked to execute its method. In effect, there is a decision made by the system that is not coded anywhere.³ Furthermore, because all of these decisions are made at run time, it is possible to send a message to an object that it does not understand (i.e., the object does not have a corresponding method). This can cause a run time error that, if not programmed to handle correctly, can cause the system to abort.

Finally, if the methods are not semantically consistent, the developer cannot assume that all methods with the same name will perform the same generic operation. For example, imagine that you have an array of type person that contains instances of employees and customers. Both of these implement a “compute pay” method. An object can send the message to each instance contained in the array to execute the compute pay method for that individual instance. In the case of an instance of employee, the compute pay method computes the amount that the employee is owed by the firm, while the compute pay method associated with instances of customers computes the amount owed the firm by the customer. Depending on whether the instance is an employee or a customer, a different meaning is associated with the method. As such, the semantics of each method must be determined individually. This substantially increases the difficulty of understanding individual objects. The key to controlling the difficulty of understanding object-oriented systems when using polymorphism is to ensure that all methods with the same name implement that same generic operation (i.e., they are semantically consistent).

Inheritance

Inheritance allows developers to define classes incrementally by reusing classes defined previously as the basis for new classes. Although we could define each class separately, it might be simpler to define one general superclass that contains the data and methods needed by the subclasses, and then have these classes inherit the properties of the superclass. Subclasses inherit the appropriate attributes and methods from the superclasses “above” them. Inheritance makes it simpler to define classes.

There have been many different types of inheritance mechanisms associated with OO systems.⁴ The most common inheritance mechanisms include different forms of single and multiple inheritance. *Single inheritance* allows a subclass to have only a single parent class. Currently, all object-oriented methodologies, databases, and programming languages permit extending the definition of the superclass through single inheritance.

³ From a practical perspective, there is an implied case statement. The system chooses the method based on the type of object being asked to execute it.

⁴ See, for example, M. Lenzerini, D. Nardi, and M. Simi, *Inheritance Hierarchies in Knowledge Representation and Programming Languages* (New York: Wiley, 1991).

Some object-oriented methodologies, databases, and programming languages allow a subclass to redefine some or all of the attributes and/or methods of its superclass. With *redefinition* capabilities, it is possible to introduce an *inheritance conflict* (i.e., an attribute [or method] of a subclass with the same name as an attribute [or method] of a superclass). For example in Figure 10-2, Doctor is a subclass of Employee. Both have methods named `computePay()`. This causes an inheritance conflict. Furthermore, when the definition of a superclass is modified, all of its subclasses are affected. This may introduce additional inheritance conflicts in one (or more) of the superclass's subclasses. For example in Figure 10-2, Employee could be modified to include an additional method, `updateSchedule()`. This would add another inheritance conflict between Employee and Doctor. Therefore, developers must be aware of the effects of the modification not only in the superclass, but also in each subclass that inherits the modification.

Finally, through redefinition capabilities, it is possible for a programmer to arbitrarily cancel the inheritance of methods by placing stubs⁵ in the subclass that will override the definition of the inherited method. If the cancellation of methods is necessary for the correct definition of the subclass, then it is likely that the subclass has been misclassified (i.e., it is inheriting from the wrong superclass).

As you can see, from a design perspective, inheritance conflicts and redefinition can cause all kinds of problems with interpreting the final design and implementation.⁶ However, most inheritance conflicts are due to poor classification of the subclass in the inheritance hierarchy (i.e., the generalization A-Kind-Of semantics are violated), or the actual inheritance mechanism violates the encapsulation principle (i.e., subclasses are capable of directly addressing the attributes or methods of a superclass). To address these issues, Jim Rumbaugh, and his colleagues, suggested the following guidelines:⁷

- Do not redefine query operations.
- Methods that redefine inherited ones should only restrict the semantics of the inherited ones.
- The underlying semantics of the inherited method should never be changed.
- The signature (argument list) of the inherited method should never be changed.

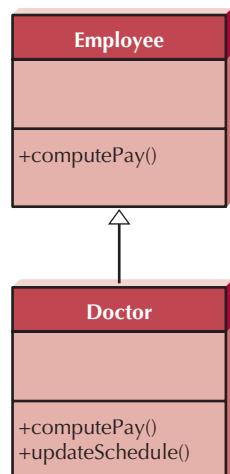


FIGURE 10-2
Single Inheritance Conflicts

However, many existing object-oriented programming languages violate these guidelines. When it comes to implementing the design, different object-oriented programming languages address inheritance conflicts differently. Therefore, it is important at this point in the development of the system to know what the programming language that you are going to use supports.

When considering the interaction of inheritance with polymorphism and dynamic binding, object-oriented systems provide the developer with a very powerful, but dangerous, set of tools. Depending on the object-oriented programming language used, this interaction can allow the same object to be associated with different classes at different times. For example, salaried employee, such as a doctor, can be treated as a generic employee, or any of its

⁵ In this case, a stub is simply the minimal definition of a method to prevent syntax errors occurring.

⁶ For more information, see Ronald J. Brachman, "I Lied about the Trees Or, Defaults and Definitions in Knowledge Representation," *AI Magazine*, 5(3) (Fall 1985), pp. 80-93.

⁷ J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design* (Englewood Cliffs, NJ: Prentice Hall, 1991).

direct and indirect superclasses. Therefore, depending on whether static or dynamic binding is supported, the same object may execute different implementations of the same method at different times. Or, if the method only is defined with the salaried employee class, and it is currently treated as a generic employee, the instance may cause a run time error to occur.⁸ As stated previously, it is important to know what object-oriented programming language is going to be used so that these kinds of issues can be solved with the design, instead of the implementation, of the class.

With *multiple inheritance*, a subclass may inherit from more than one superclass. In this situation, the types of inheritance conflicts are multiplied. In addition to the possibility of having an inheritance conflict between the subclass and one (or more) of its superclasses, it is now possible to have conflicts between two (or more) superclasses. In this latter case, there are three different types of additional inheritance conflicts that can occur:

1. Two inherited attributes (or methods) have the same name and semantics.
2. Two inherited attributes (or methods) have different names but with identical semantics (i.e., they are *synonyms*).
3. Two inherited attributes (or methods) have the same name but with different semantics (i.e., they are *homonyms*). This also violates the proper use of polymorphism.

For example, in Figure 10-3, Robot-Employee is a subclass of both Employee and Robot. In this case, Employee and Robot conflict with the attribute name. Which one should Robot-Employee inherit? Because they are the same, semantically speaking, does it really matter? It is also possible that Employee and Robot could have a semantic conflict on the classification and type attributes if they are synonyms. Practically speaking, the only way to prevent this situation is for the developer to catch it during the design of the subclass. Finally, what if the runningTime attributes are homonyms? In the case Employee objects, the runningTime attribute stores the employee's time running a mile, while the runningTime attribute for Robot objects store the average time between checkups. Should Robot-Employee inherit both of them? It really depends on whether the robot employees can run the mile or not. With the potential for these additional types of conflicts, there is a risk of decreasing the understandability in an object-oriented system, instead of increasing it, through the use of multiple inheritance. As such, great care should be taken when using multiple inheritance.

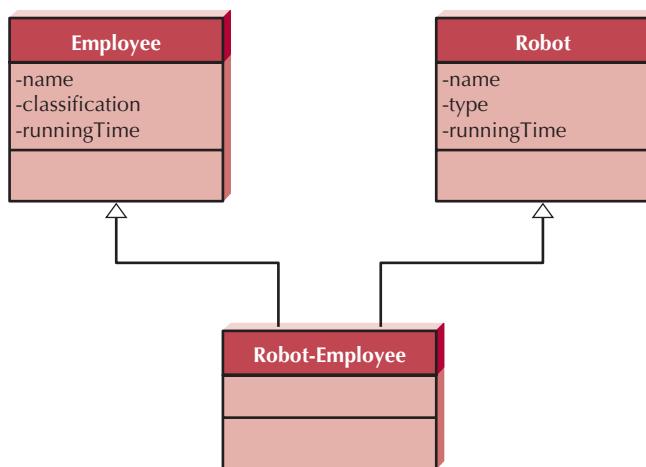


FIGURE 10-3
Additional Inheritance Conflicts with Multiple Inheritance

⁸ This happens with novices quite regularly when using C++.

CONCEPTS

10-A Inheritance Abuses

IN ACTION

Meilir Page-Jones, through his consulting company, identified a set of abuses of inheritance. In some cases, these abuses led to lengthy and bloody disputes, gruesome implementations, and in one case, it led to the destruction of the development team. In all cases, the error was in not enforcing a generalization (A-Kind-Of) semantics. In one case, the inheritance hierarchy was inverted: BoardMember was a superclass of Manager which was a superclass of Employee. However, in this case, an Employee is NOT A-Kind-Of Manager which is NOT A-Kind-Of BoardMember. In fact, the opposite was true. However, if you think of an Organization Chart, a BoardMember is “super” to a Manager, which is “super” to an Employee. In another

example, the client’s firm attempted to use inheritance to model a membership idea (e.g., Student is a member of a club). However, the club should have had an attribute that contained the student members. In the other examples, inheritance was used to implement an association relationship and an aggregation relationship.

Source: Meilir Page-Jones, *Fundamentals of Object-Oriented Design in UML* (Reading, MA: Addison-Wesley, 2000).

Question:

1. As an analyst, how can you attempt to avoid these types of inheritance abuses?

DESIGN CRITERIA

When considering the design of an object-oriented system, there is a set of criteria that can be used to determine whether the design is a good one or a bad one. According to Coad and Yourdon,⁹ “A good design is one that balances trade-offs to minimize the total cost of the system over its entire lifetime.” These criteria include coupling, cohesion, and connascence.

Coupling

Coupling refers to how interdependent or interrelated the modules (classes, objects, and methods) are in a system. The higher the interdependency, the more likely changes in part of a design can cause changes to be required in other parts of the design. For object-oriented systems, Coad and Yourdon¹⁰ identified two types of coupling to consider: interaction and inheritance.

Interaction coupling deals with the coupling among methods and objects through message passing. Lieberherr and Holland put forth the *Law of Demeter* as a guideline to minimize this type of coupling.¹¹ Essentially, the law minimizes the number of objects that can receive messages from a given object. The law states that an object should only send messages to one of the following: itself, an object that is contained in an attribute of the object (or one of its superclasses), an object that is passed as a parameter to the method, an object that is created by the method, or an object that is stored in a global variable. In each case, interaction coupling is increased. For example, the coupling increases between the objects if the calling method passes attributes to the called method, or if the calling method depends on the value being returned by the called method.

There are six types of interaction coupling, each falling on different parts of a good-to-bad continuum. They range from no direct coupling (Good) up to content coupling (Bad).

⁹ Peter Coad and Edward Yourdon, *Object-Oriented Design* (Englewood Cliffs, NJ: Yourdon Press, 1991), p. 128.

¹⁰ *Ibid.*

¹¹ Karl J. Lieberherr and Ian M. Holland, “Assuring Good Style for Object-Oriented Programs,” *IEEE Software*, 6 (5) (September, 1989), pp. 38-48 and Karl J. Lieberherr, *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns* (Boston, MA: PWS Publishing, 1996).

Figure 10-4 presents the different types of interaction coupling. In general, interaction coupling should be minimized. The one possible exception is that non-problem domain classes must be coupled to their corresponding problem domain classes. For example, a report object (on the human computer interaction layer) that displays the contents of an employee object (on the problem domain layer) will be dependent on the employee object. In this case, for optimization purposes, the report class may be even content or pathologically coupled to the employee class. However, problem domain classes should never be coupled to non-problem domain classes.

Inheritance coupling, as its name implies, deals with how tightly coupled the classes are in an inheritance hierarchy. Most authors tend to say simply that this type of coupling is desirable. However, depending on the issues raised previously with inheritance—inheritance conflicts, redefinition capabilities, and dynamic binding—a high level of inheritance coupling may not be a good thing. For example, should a method defined in a subclass be allowed to call a method defined in one of its superclasses? Or, should a method defined in a subclass refer to an attribute defined in one of its superclasses? Or, even more confusing, can a method defined in an abstract superclass depend on its subclasses to define a method or attribute on which it is dependent? Depending on the object-oriented programming language being used, all of these options are possible.

As Snyder has pointed out, most problems with inheritance is the ability within the object-oriented programming languages to violate the encapsulation and information hiding principles.¹² Therefore again, knowledge of which object-oriented programming language is to be used is crucial. From a design perspective, the developer will need to optimize

Level	Type	Description
Good	No Direct Coupling	The methods do not relate to one another; that is, they do not call one another.
	Data	The calling method passes a variable to the called method. If the variable is composite, (i.e., an object), the entire object is used by the called method to perform its function.
	Stamp	The calling method passes a composite variable (i.e., an object) to the called method, but the called method only uses a portion of the object to perform its function.
	Control	The calling method passes a control variable whose value will control the execution of the called method.
	Common or Global	The methods refer to a “global data area” that is outside the individual objects.
Bad	Content or Pathological	A method of one object refers to the inside (hidden parts) of another object. This violates the principles of encapsulation and information hiding. However, C++ allows this to take place through the use of “friends.”

Source: These types were adapted from Page-Jones, *The Practical Guide to Structured Systems Design*, 2nd Ed, Englewood Cliffs, NJ: Yarden Press, 1988; and Glenford Myers, *Composite/Structured Design*. New York: Van Nostrand Reinhold, 1978.

FIGURE 10-4
Types of Interaction Coupling

¹² Alan Snyder, “Encapsulation and Inheritance in Object-Oriented Programming Languages,” in: N. Meyrowitz, Ed., *OOPSLA '86 Conference Proceedings, ACM SigPlan Notices*, 21 (11) (November 1986) and Alan Snyder, “Inheritance and the Development of Encapsulated Software Components,” in: B. Shriver and P. Wegner, Eds., *Research Directions in Object-Oriented Programming* (Cambridge, MA: MIT Press, 1987).

the trade-offs of violating the encapsulation and information hiding principles and increasing the desirable coupling between subclasses and its superclasses. The best way to solve this conundrum is to ensure that inheritance is used only to support generalization/specialization (A-Kind-Of) semantics and the principle of substitutability (see Chapter 7). All other uses should be avoided.

Cohesion

Cohesion refers to how single-minded a module (class, object, or method) is within a system. A class or object should only represent one thing, and a method should only solve a single task. Three general types of cohesion: method, class, and generalization/specialization, have been identified by Coad and Yourdon¹³ for object-oriented systems. Each of these is described below.

Method cohesion addresses the cohesion within an individual method (i.e., how single-minded is a method). Methods should do one and only one thing. A method that actually performs multiple functions is more difficult to understand than one that only performs a single function. There are seven types of method cohesion that have been identified (see Figure 10-5). They range from functional cohesion (Good) down to coincidental cohesion (Bad). In general, method cohesion should be maximized.

Level	Type	Description
Good	Functional	A method performs a single problem-related task (e.g., Calculate current GPA).
	Sequential	The method combines two functions in which the output from the first one is used as the input to the second one (e.g., format and validate current GPA).
	Communicational	The method combines two functions that use the same attributes to execute (e.g., calculate current and cumulative GPA).
	Procedural	The method supports multiple weakly related functions. For example, the method could calculate student GPA, print student record, calculate cumulative GPA, and print cumulative GPA.
	Temporal or Classical	The method supports multiple related functions in time (e.g., initialize all attributes).
	Logical	The method supports multiple related functions, but the choice of the specific function is chosen based on a control variable that is passed into the method. For example, the called method could open a checking account, open a savings account, or calculate a loan, depending on the message that is sent by its calling method.
Bad	Coincidental	The purpose of the method cannot be defined or it performs multiple functions that are unrelated to one another. For example, the method could update customer records, calculate loan payments, print exception reports, and analyze competitor pricing structure.

Source: These types were adapted from Page-Jones, *The Practical Guide to Structured Systems* and Myers *Composite/Structured Design*.

FIGURE 10-5
Types of Method Cohesion

¹³ Coad and Yourdon, *Object-Oriented Design*.

Class cohesion is the level of cohesion among the attributes and methods of a class (i.e., how single-minded is a class). A class should only represent one thing, such as an employee, a department, or an order. All attributes and methods contained in a class should be required for the class to represent the thing. For example, an employee class should have attributes that deal with a social security number, last name, first names, middle initial, addresses, and benefits, but it should not have attributes like door, engine, or hood. Furthermore, there should be no attributes or methods that are never used. In other words, a class should have only the attributes and methods necessary to fully define instances for the problem at hand. In this case, we have *ideal class cohesion*. Glenford Meyers suggested that a cohesive class¹⁴ should:

- contain multiple methods that are visible outside of the class and that each visible method only performs a single function (i.e., it has functional cohesion),
- have methods that only refer to attributes or other methods defined with the class or its superclass(es), and
- not have any control-flow couplings between its methods.

Page-Jones¹⁵ has identified three less than desirable types of class cohesion: mixed-instance, mixed-domain, and mixed-role (see Figure 10-6). An individual class can have a mixture of any of the three types.

Generalization/specialization cohesion addresses the sensibility of the inheritance hierarchy. How are the classes in the inheritance hierarchy related? Are the classes related through a generalization/specialization (A-Kind-Of) semantics? Or, are they related via some association, aggregation, or membership type of relationship that was created for simple reuse purposes? Recall, all of the issues raised previously on the use of inheritance. Highly cohesive inheritance hierarchies only should support the semantics of generalization and specialization (A-Kind-Of) and the principle of substitutability.

Level	Type	Description
Good	Ideal	The class has none of the mixed cohensions.
	Mixed-Role	The class has one or more attributes that relate objects of the class to other objects on the same layer (e.g., the problem domain layer), but the attribute(s) have nothing to do with the underlying semantics of the class.
	Mixed-Domain	The class has one or more attributes that relate objects of the class to other objects on a different layer. As such, they have nothing to do with the underlying semantics of the thing that the class represents. In these cases, the offending attribute(s) belongs in another class located on one of the other layers. For example, a port attribute located in a problem domain class should be in a system architecture class that is related to the problem domain class.
Worse	Mixed-Instance	The class represents two different types of objects. The class should be decomposed into two separate classes. Typically, different instances only use a portion of the full definition of the class.

Source: Page-Jones, *Fundamentals of Object-Oriented Design in UML*.

FIGURE 10-6
Types of Class Cohesion

¹⁴ We have adapted his informational-strength module criteria from structured design to object-oriented design. (see Glenford J. Myers, *Composite/Structured Design* [New York, NY: Van Nostrand Reinhold, 1978]).

¹⁵ See Meilir Page-Jones, *Fundamentals of Object-Oriented Design in UML* (Reading, MA: Addison-Wesley, 2000).

Connascence

*Connascence*¹⁶ generalizes the ideas of cohesion and coupling, and it combines them with the arguments for encapsulation. To accomplish this, three levels of encapsulation have been identified. Level-0 encapsulation refers to the amount of encapsulation realized in an individual line of code, Level-1 encapsulation is the level of encapsulation attained by combining lines of code into a method, and Level-2 encapsulation is achieved by creating classes that contain both methods and attributes. Method cohesion and interaction coupling primarily address level-1 encapsulation. Class cohesion, generalization/specialization cohesion, and inheritance coupling only address level-2 encapsulation. Connascence, as a generalization of cohesion and coupling, addresses both level-1 and level-2 encapsulation.

But what exactly is connascence? Connascence literally means to be born together. From an object-oriented design perspective, it really means that two modules (classes or methods) are so intertwined, that if you make a change in one, it is likely that a change in the other will be required. On the surface, this is very similar to coupling, and as such should be minimized. However, when you combine it with the encapsulation levels, it is not quite as simple as that. In this case, you want to (1) minimize overall connascence by eliminating any unnecessary connascence throughout the system, (2) minimize connascence across any encapsulation boundaries, such as method boundaries and class boundaries, and (3) maximize connascence within any encapsulation boundary.

Based on these guidelines, a subclass should never directly access any hidden attribute or method of a superclass (i.e., a subclass should not have special rights to the properties of its superclass(es)). If direct access to the non-visible attributes and methods of a superclass by its subclass is allowed, which is permitted in most object-oriented programming languages, and a modification to the superclass is made, then due to the connascence between the subclass and its superclass, it is likely that a modification to the subclass also will be required. In other words, the subclass has access to something across an encapsulation boundary (the class boundary between the subclass and the superclass). Practically speaking, you should maximize the cohesion (connascence) within an encapsulation boundary and minimize the coupling (connascence) between the encapsulation boundaries. There are many possible types of connascence. Figure 10-7 describes five of the types.

OBJECT DESIGN ACTIVITIES

The design activities for classes and methods are really an extension of the analysis and evolution activities presented previously (see Chapters 6 through 9). In this case, we expand the descriptions of the partitions, layers, and classes. Practically speaking, the expanded descriptions are created through the activities that take place during the detailed design of the classes and methods. As such, we limit this discussion to the detailed design of classes and methods. The activities used to design classes and methods include additional specification of the current model, identifying opportunities for reuse, restructuring the design, optimizing the design, and finally, mapping the problem domain classes to an implementation language. Of course, any changes made to a class on one layer can cause the classes on the other layers that are coupled to it to be modified also. The object design activities are described in this section.

¹⁶ See Meilir Page-Jones, "Comparing Techniques by Means of Encapsulation and Connascence," *Communications of the ACM*, 35 (9) (September 1992), pp. 147–151.

Type	Description
Name	If a method refers to an attribute, it is tied to the name of the attribute. If the attribute's name changes, the content of the method will have to change.
Type or Class	If a class has an attribute of type A, it is tied to the type of the attribute. If the type of the attribute changes, the attribute declaration will have to change.
Convention	A class has an attribute in which a range of values has a semantic meaning (e.g., account numbers whose values range from 1000 to 1999 are assets). If the range would change, then every method that used the attribute would have to be modified.
Algorithm	Two different methods of a class are dependent on the same algorithm to execute correctly (e.g., insert an element into an array and find an element in the same array). If the underlying algorithm would change, then the insert and find methods would also have to change.
Position	The order of the code in a method or the order of the arguments to a method is critical for the method to execute correctly. If either is wrong, then the method will, at least, not function correctly.

Source: Page-Jones, "Comparing Techniques by Means of Encapsulation and Connascence" and Page-Jones, *Fundamentals of Object-Oriented Design in UML*.

FIGURE 10-7
Types of Connascence

Additional Specification

At this point in the development of the system, it is crucial to review the current set of structural and behavioral models.¹⁷ First, we should ensure that the classes on the problem domain layer are both necessary and sufficient to solve the underlying problem. To do this, we need to be sure that there are no missing attributes or methods and no extra or unused attributes or methods in each class. Furthermore, are there any missing or extra classes? If we have done our job well during the analysis phase, there will be few, if any, attributes or methods to add to the models. However, it is always better to be safe than sorry.

Second, we need to finalize the *visibility* (hidden or visible) of the attributes and methods in each class. Depending on the object-oriented programming language used, this could be predetermined (e.g., in Smalltalk attributes are hidden and methods are visible¹⁸).

Third, we need to decide on the signature of every method in every class. The *signature* of a method comprises three parts: the name of the method, the parameters or arguments that must be passed to the method, and the type of value that the method will return to the calling method.

Fourth, we need to define any constraints that must be preserved by the objects (e.g., an attribute of an object that can only have values in a certain range). There are three different types of constraints: pre-conditions, post-conditions, and invariants. These are captured in the form of contracts (described later in this chapter) and assertions added to CRC cards and class diagrams. We also must decide how to handle a violation of a constraint. Should the system simply abort? Should the system automatically undo the change that caused the violation? Should the system let the end user determine the approach to correct the violation? In other words, the designer must design the errors that the system is expected to handle. It is best to not leave these types of problems for the programmer to solve. Violations of a constraint are known as *exceptions* in languages, such as C++ and Java.

¹⁷ I know, we have mentioned this before. However, you cannot overemphasize the importance of constantly reviewing the evolving design.

¹⁸ By default, most object-oriented analysis and design approaches assume Smalltalk's approach.

The additional specification activities also are applicable to the other layers: data management Identifying Opportunities for Reuse section (Chapter 11), human computer interaction (Chapter 12), and physical architecture (Chapter 13).

Identifying Opportunities for Reuse

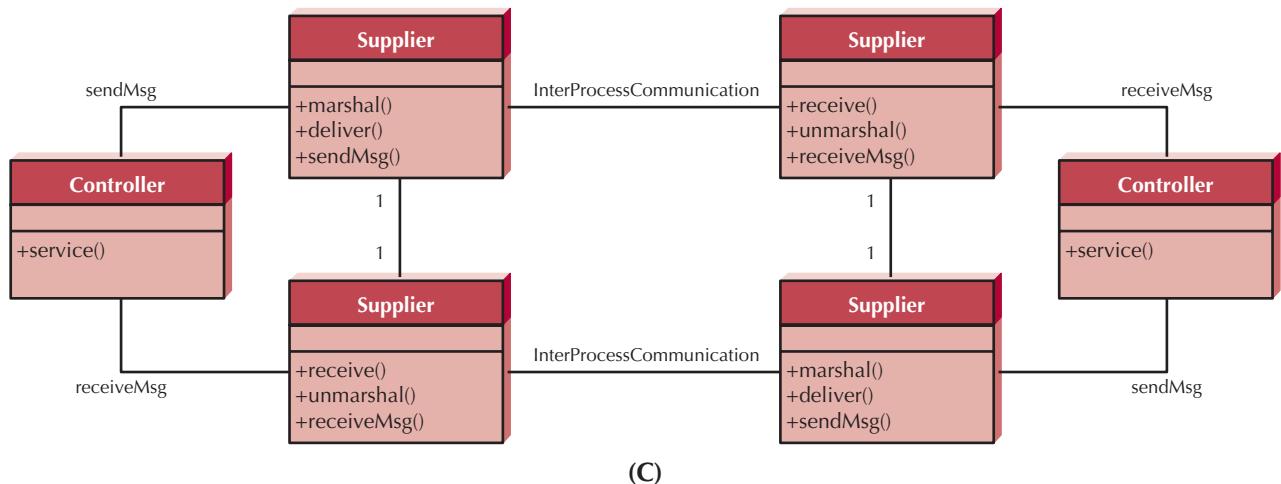
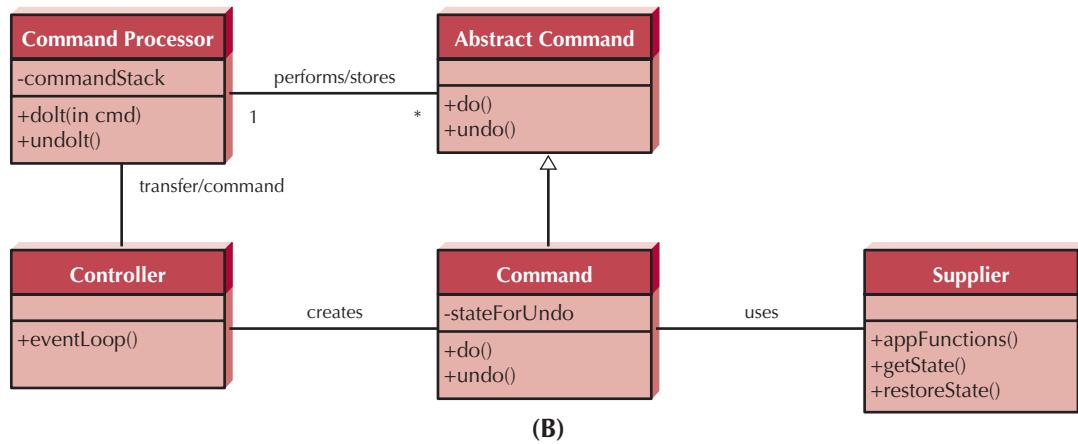
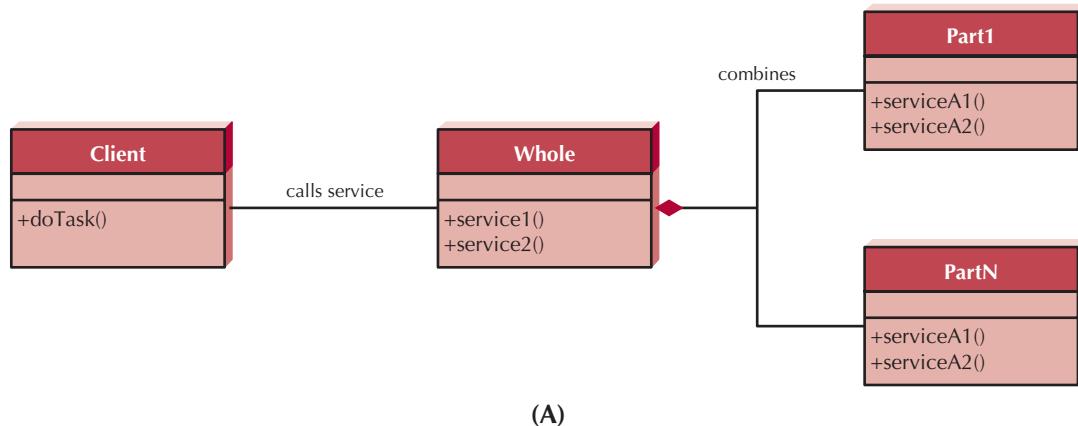
Previously, we looked at possibly utilizing reuse in our structural models in the analysis phase through the use of *patterns* (see Chapter 7). In the design phase, in addition to using analysis patterns, there are opportunities for using design patterns, frameworks, libraries, and components. The opportunities will vary depending on which layer is being reviewed. For example, it is doubtful that a class library will be of much help on the problem domain layer, but an appropriate class library could be of great help on the foundation layer. In this section, we describe the use of design patterns, frameworks, libraries, and components.

Like analysis patterns, *design patterns* are simply useful grouping of collaborating classes that provide a solution to a commonly occurring problem. The primary difference between analysis and design patterns is that design patterns are useful in solving “a general design problem in a particular context.”¹⁹ While analysis patterns tended to aid in filling out a problem domain representation. For example, a useful pattern is the Whole-Part pattern (see Figure 10-8, Part A). The Whole-Part pattern explicitly supports the Aggregation and Composition relationships within the UML. Another useful design pattern is the Command Processor pattern (see Figure 10-8, Part B). The primary purpose of the Command Processor pattern is to force the designer to explicitly separate the interface to an object (Command) from the actual execution of the operation (Supplier) behind the interface. Finally, some of the design patterns support different physical architectures (see Chapter 13). For example, the Forwarder-Receiver pattern (see Figure 10-8, Part C) supports a peer-to-peer architecture. Many design patterns are available in C++ or Java source code.

A *framework* is composed of a set of implemented classes that can be used as a basis for implementing an application. For example, there are frameworks available for CORBA and DCOM on which you could base the implementation of part of the physical architecture layer. Most frameworks allow you to create subclasses to inherit from classes in the framework. There are object-persistence frameworks that can be purchased and used to add persistence to the problem domain classes, which would be helpful on the data management layer. Of course, when inheriting from classes in a framework, you are creating a dependency (i.e., increasing the inheritance coupling from the subclass to the superclass). Therefore, if you use a framework, and the vendor makes changes to the framework, you will have to at least recompile the system when you upgrade to the new version of the framework.

A *class library* is similar to a framework in that you typically have a set of implemented classes that were designed for reuse. However, frameworks tend to be more domain-specific. In fact, frameworks may be built using a class library. A typical class library could be purchased to support numerical or statistical processing, file management (data management layer), or user interface development (human computer interaction layer). In some cases, you will create instances of classes contained in the class library and in other cases you will extend classes in the class library by creating subclasses based on them. Like frameworks, if you use inheritance to reuse the classes in the class library, you will run into all of the issues dealing with inheritance coupling and connascence. If you directly instantiate classes in the class library, you will create a dependency between your object and the library object based on the signatures of the methods in the library object. This will increase the interaction coupling between the class library object and your object.

¹⁹ Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Reading, MA: Addison-Wesley, 1995).

**FIGURE 10-8** Sample Design Patterns²⁰

²⁰ These design patterns, along with many more, can be found in Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal, *Pattern-Oriented Software Architecture: A System of Patterns* (Chichester, UK: Wiley, 1996).

Source: Buschmann, et al, *Pattern-Oriented Software Architecture: A System of Patterns* (John Wiley & Sons: 2000).

A *component* is a self-contained, encapsulated piece of software that can be “plugged” into a system to provide a specific set of required functionality. Today, there are many components available for purchase that have been implemented using *ActiveX* or *JavaBean* technologies. A component has a well-defined *Application Program Interface* (API). The API is essentially a set of method interfaces to the objects contained in the component. The internal workings of the component are hidden behind the API. Components can be implemented using class libraries and frameworks. However, components also can be used to implement frameworks. Unless, the API changes between versions of the component, upgrading to a new version normally will require only linking the component back into the application. As such, recompilation typically is not required.

Which of these approaches should you use? It depends on what you are trying to build. In general, frameworks are used mostly to aid in developing objects on the physical architecture, human computer interaction, or data management layers; components are used primarily to simplify the development of objects on the problem domain and human computer interaction layers; and class libraries are used to develop frameworks and components and to support the foundation layer.

Restructuring the Design

Once the individual classes and methods have been specified, and the class libraries, frameworks, and components have been incorporated into the evolving design, you should use factoring to restructure the design. *Factoring*, if you recall from Chapter 9, is the process of separating out aspects of a method or class into a new method or class to simplify the overall design. For example, when reviewing a set of classes on a particular layer, you might discover that a subset of them shares a similar definition. In that case, it may be useful to factor out the similarities and create a new class. Based on the issues related to cohesion, coupling, and connascence, the new class may be related to the old classes via inheritance (generalization) or through an aggregation or association relationship.

Another process that is useful to restructure the evolving design is *normalization*. Normalization is described in Chapter 11 in relation to relational databases. However, normalization can be useful at times to identify potential classes that are missing from the design. Also, related to normalization is the requirement to implement the actual association and aggregation relationships as attributes. Virtually no object-oriented programming language differentiates between attributes and association and aggregation relationships. Therefore, all association and aggregation relationships must be converted to attributes in the classes.

Finally, all inheritance relationships should be challenged to ensure that they only support a generalization/specialization (A-Kind-Of) semantics and the principle of substitutability. Otherwise, all of the problems mentioned previously with inheritance coupling, class cohesion, and generalization/ specialization cohesion will come to pass.

YOUR

TURN

10-1 Campus Housing

In the previous chapters, you have been working on a system for the campus housing service. Based on the current set of functional, structural, and behavioral models that you have developed, are there potential

opportunities of reuse in developing the system? Search the Web for potential patterns, class libraries, and components that could be useful in developing this system.

Optimizing the Design²¹

Up until now, we have focused our energy on developing an understandable design. With all of the classes, patterns, collaborations, partitions, and layers designed and with all of the class libraries, frameworks, and components included in the design, understandability has been, as it should have been, our primary focus. However, increasing the understandability of a design typically creates an inefficient design. And conversely, focusing on efficiency issues will deliver a design that is more difficult to understand. A good practical design will manage the inevitable tradeoffs that must occur to create an acceptable system. In this section, we describe a set of simple optimizations that can be used to create a more efficient design.

The first optimization to consider is to review the access paths between objects. In some cases, a message from one object to another may have a long path to traverse (i.e., it goes through many objects). If the path is long, and the message is sent frequently, a redundant path should be considered. Adding an attribute to the calling object that will store a direct connection to the object at the end of the path can accomplish this.

A second optimization is to review each attribute of each class. Which methods use the attributes and which objects use the methods should be determined. If the only methods that use an attribute are read and update methods, and only instances of a single class send messages to read and update the attribute, then the attribute may belong with the calling class instead of the called class. Moving the attribute to the calling class will substantially speed up the system.

A third optimization is to review the direct and indirect fan-out of each method. *Fan-out* refers to the number of messages sent by a method. The direct fan-out is the number of messages sent by the method itself, while the indirect fan-out also includes the number of messages sent by the methods called by the other methods in a message tree. If the fan-out of a method is high relative to the other methods in the system, the method should be optimized. One way to do this is to consider adding an index (see Chapter 11) to the attributes used to send the messages to the objects in the message tree.

A fourth optimization is to look at the execution order of the statements in often-used methods. In some cases, it may be possible to rearrange some of the statements to be more efficient. For example, if it is known, based on the objects in the system, that a search routine can be narrowed by searching on one attribute before another one, then the search algorithm should be optimized by forcing it to always search in a pre-defined order.

A fifth optimization is to avoid recomputation by creating a *derived attribute* (or *active value*) (e.g., a total that will store the value of the computation). This is also known as caching computational results. This can be accomplished by adding a *trigger* to the attributes contained in the computation (i.e., those attributes on which the derived attribute is dependent). This would require a recomputation to take place only when one of the attributes that go into the computation is changed. Another approach would be to mark the derived attribute, and delay the recomputation until the next time the derived attribute is accessed. This last approach delays the recomputation as long as possible. In this manner, a computation does not occur unless it has to occur. Otherwise, every time a derived attribute needs to be accessed, a computation would be required.

²¹ The material contained in this section is based on James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, *Object-Oriented Modeling and Design* (Englewood Cliffs, NJ: Prentice-Hall, 1991), and Bernd Brugge and Allen H. Dutoit, *Object-Oriented Software Engineering: Conquering Complex and Changing Systems* (Englewood Cliffs, NJ: Prentice Hall, 2000).

**YOUR
TURN**

10-2 Campus Housing

Assume that you are the project leader for the campus housing system that you have been developing over the previous chapters and that you just modified in Your Turn 10-1. However, as you review the current set of models, you realize that even though the models provide a rather complete description of the problem domain layer, the evolving models have begun to become unmanageable.

And as project leader, you also need to guarantee that the design will be efficient. Create a set of discussion points that you will use to explain to your development team the importance of optimizing the design before jumping into coding. Be sure to include an example of each optimization technique that can be used in the current set of models for the campus housing system.

Mapping Problem Domain Classes to Implementation Languages²²

Up until this point in the development of the system, it has been assumed that the classes and methods in the models would be implemented directly in an object-oriented programming language. However, now it is important to map the current design to the capabilities of the programming language used. For example, if you have used multiple inheritance in your design, but you are implementing in a language that only supports single inheritance, then the multiple inheritance must be factored out of the design. Or, if the implementation is to be done in an object-based language, one that does not support inheritance,²³ or a non-object-based language, such as C or Pascal, we must map the problem domain objects to programming constructs that can be implemented using the chosen implementation environment. In this section, we describe a set of rules that can be used to do the necessary mapping.

Implementing Problem Domain Classes in a Single-Inheritance Language The only issue associated with implementing problem domain objects is the factoring out of any multiple inheritance, that is, the use of more than one superclass, used in the evolving design. For example, if you were to implement the solution in Java, Smalltalk, or Visual Basic.net, you must factor out any multiple inheritance. The easiest way to do this is to use the following rule:

RULE 1a: Convert the additional inheritance relationships to association relationships. The multiplicity of the new association from the subclass to the “superclass” should be 1..1. If the additional superclasses are concrete, that is, they can be instantiated themselves, then the multiplicity from the superclass to the subclass is 0..1. Otherwise, it is 1..1. Furthermore, an exclusive-or (XOR) constraint must be added between the associations. Finally, you must add appropriate methods to ensure that all information is still available to the original class.

OR

RULE 1b: Flatten the inheritance hierarchy by copying the attributes and methods of the additional superclass(es) down to all of the subclasses and remove the additional superclass from the design.²⁴

²² The mapping rules presented in this section are based on material in Coad and Yourdon, *Object-Oriented Design*.

²³ In this case, we are talking about implementation inheritance, not the so-called interface inheritance. Interface inheritance supported by Visual Basic and Java only supports inheriting the requirements to implement certain methods, not any implementation. Java and Visual Basic.net also support single inheritance as described in this text, whereas Visual Basic 6 only supports interface inheritance.

²⁴ It is also a good idea to document this modification in the design so that in the future, modifications to the design can be maintained easily.

Figure 10-9 demonstrates the application of the above rules. Part A of Figure 10-9 portrays a simple example of multiple inheritance where Class 1 inherits from both SuperClass1 and SuperClass2, and Class 2 inherits from both SuperClass2 and SuperClass3. Assuming that SuperClass2 is concrete, we apply Rule 1a to Part A, and we end up with the diagram in Part B where we have added the association between Class1 and SuperClass2 and the association between Class2 and SuperClass2. Furthermore, the multiplicities have been added correctly, and the XOR constraint has been applied. If we apply Rule 1b to Part A, we end up with the diagram in Part C where all of the attributes of SuperClass2 have been copied down into Class1 and Class2. In this latter case, you may have to deal with the effects of inheritance conflicts (see earlier in the chapter).

The advantage of Rule 1a is that all problem domain classes identified during analysis are preserved. This allows for maximum flexibility of maintenance of the design of the problem domain layer. However, Rule 1a increases the amount of message passing required in the system, and it has added processing requirements involving the XOR constraint, thus reducing the overall efficiency of the design. As such, our recommendation is to limit Rule 1a to only be applied when dealing with “extra” superclasses that are concrete because they have an independent existence in the problem domain. Use Rule 1b when they are abstract because they do not have an independent existence from the subclass.

Implementing Problem Domain Objects in a Object-Based Language If we are going to implement our solution in an *object-based language* (i.e., a language that supports the creation of object, but does not support implementation inheritance), we must factor out all uses of inheritance from the problem domain class design. For example, if you were to implement your design in Visual Basic 6 or earlier, you would have to remove all uses of inheritance in the design. Applying the above rule to all superclasses will enable you to restructure your design without any inheritance.

Figure 10-10 demonstrates the application of the above rules. Part A of Figure 10-10 shows the same simple example of multiple inheritance portrayed in Figure 10-9 where Class1 inherits from both SuperClass1 and SuperClass2, and Class2 inherits from both SuperClass2 and SuperClass3. Assuming that the superclasses are concrete, we apply Rule 1a to Part A and we end up with the diagram in Part B where we have added the associations, the multiplicities, and the XOR constraint. If we apply Rule 1b to Part A and we assume that the superclasses are abstract, we end up with the diagram in Part C where all of the attributes of the superclasses have been copied down into Class1 and Class2. In this latter case, you may have to deal with the effects of inheritance conflicts (see earlier in the chapter). What would you recommend to do if SuperClass 2 was concrete and SuperClass1 and SuperClass3 were abstract.

Implementing Problem Domain Objects in a Traditional Language From a practical perspective, you are much better off implementing an object-oriented design in an object-oriented programming language, such as C++, Java, Smalltalk, or Visual Basic.net. However, implementing an object-oriented design in an object-based language, such as Visual

YOUR TURN

10-3 Dentist Office Appointment System

In the previous chapters, we have been using a dentist office appointment system as an example. Assume that you now know that the system must be implemented in Visual

Basic 6, which does not support implementation inheritance. As such, redraw the class diagram factoring out the use of inheritance in the design by applying the above rules.

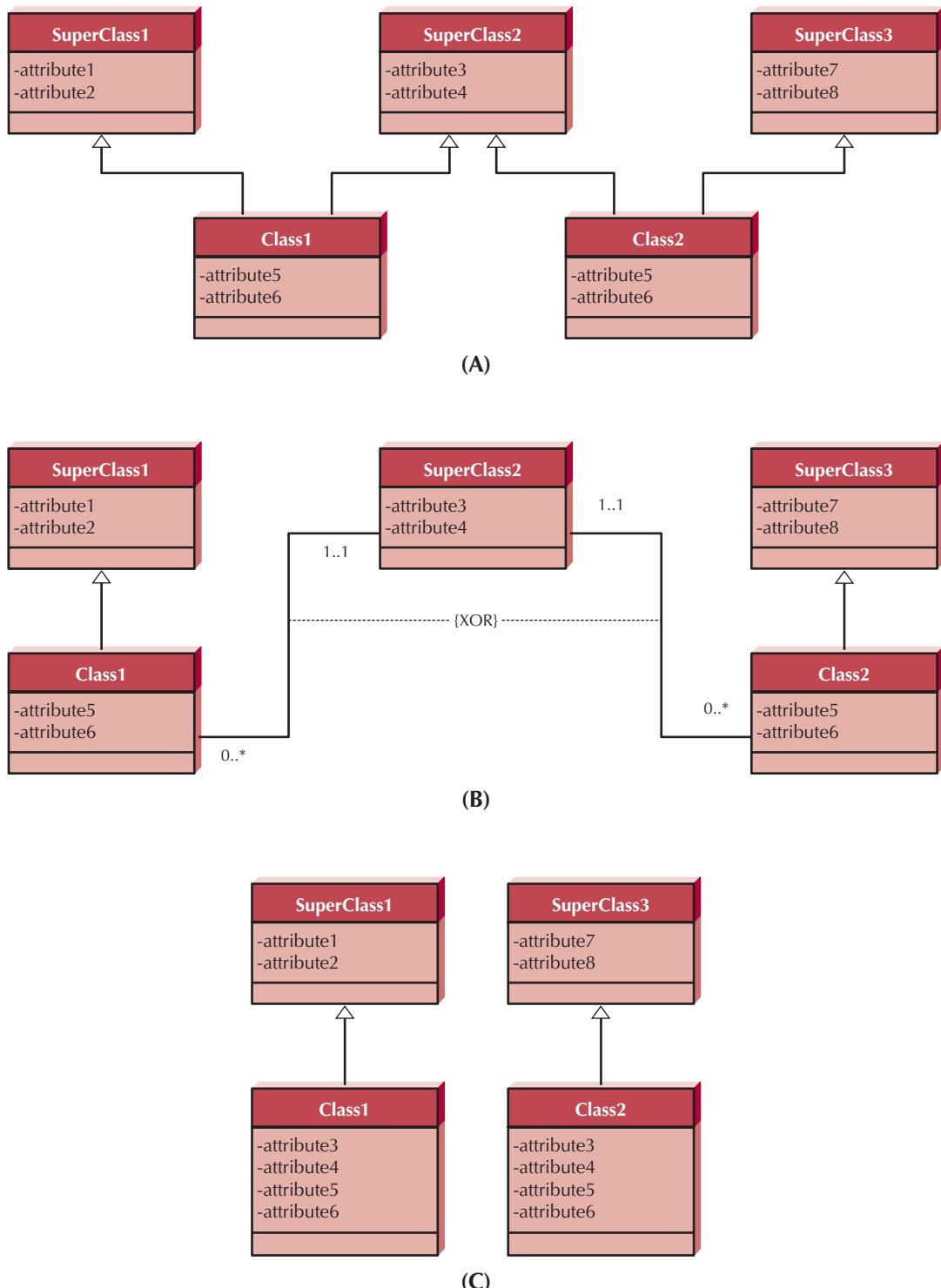


FIGURE 10-9 Factoring Out Multiple Inheritance Effect for a Single-Inheritance Language

Basic 6, is preferable to attempting to implement it in a traditional programming language, such as C or Cobol. Practically speaking, the “gulf” between an object-oriented design and a traditional programming language is simply too great for mere mortals to be able to cross. The best advice that we can give to you on implementing an object-oriented design

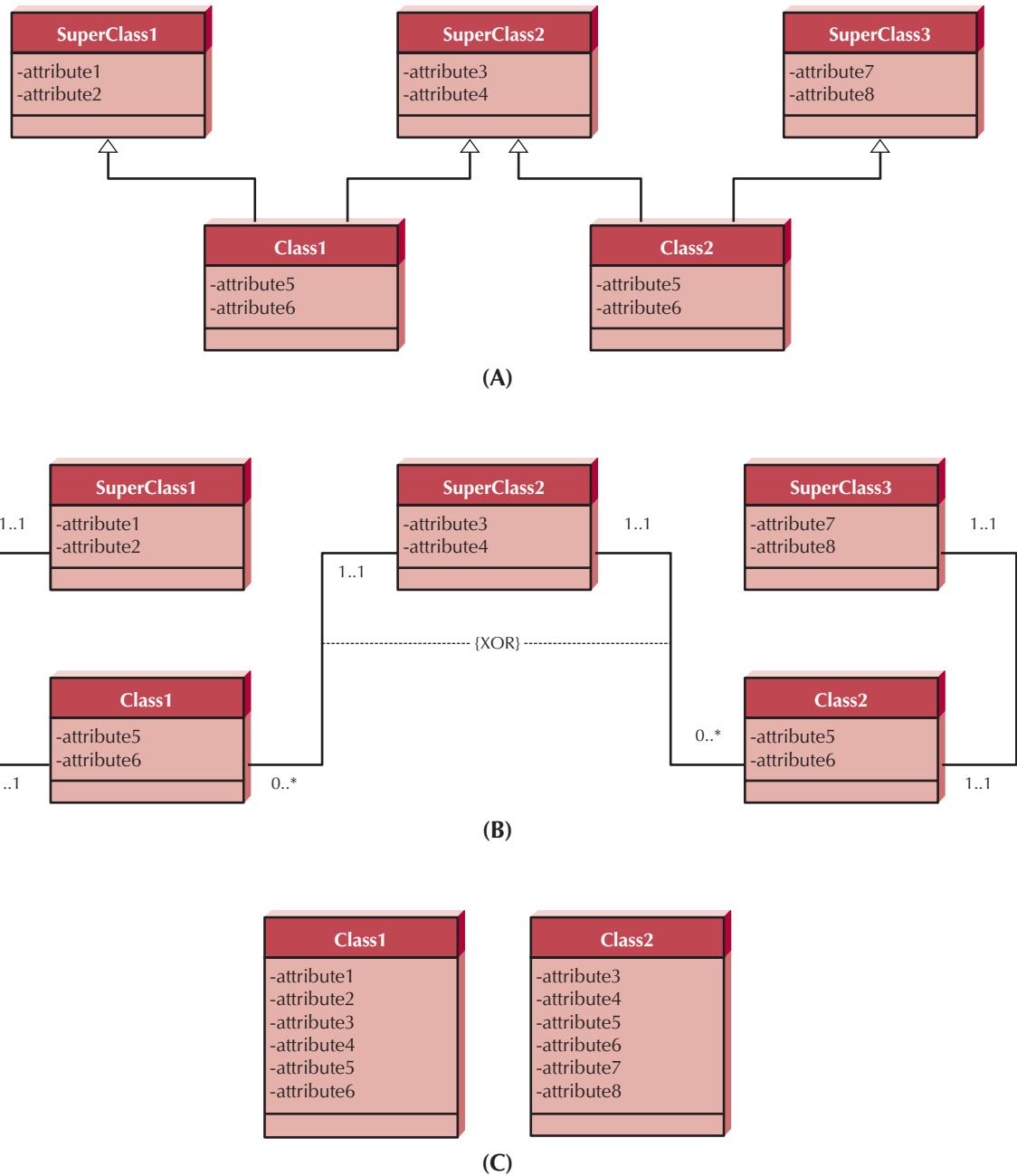


FIGURE 10-10 Factoring Out Multiple Inheritance Effect for an Object-Based Language

in a traditional programming language, is to run away as fast and as far as possible from the project. However, if you are brave (foolish?) enough to attempt this, you must realize that, in addition to factoring out inheritance from the design, you will have to factor out all uses of polymorphism, dynamic binding, encapsulation, and information hiding. To say the least, this is quite a bit of additional work to be accomplished. Furthermore, the manner in which you factor these object-oriented features out of the detailed design of the system tends to be language dependent. As such, this is beyond the scope of this text.

CONSTRAINTS AND CONTRACTS

Contracts were introduced in Chapter 7 in association with collaborations. If you recall, a *contract* formalizes the interactions between the client and server objects, where a *client (consumer)* object is an instance of a class that sends a message to a *server (supplier)* object that executes one of its methods in response to the request. Contracts are modeled on the legal notion of a contract where both parties, client and server objects, have obligations and rights. Practically speaking, a contract is a set of constraints and guarantees. If the constraints are met, then the server object will guarantee certain behavior.²⁵ Constraints can be written in either a natural language (e.g., English, Structured English, pseudocode) or a formal language (e.g., UML's Object Constraint Language²⁶).

Types of Constraints

There are three different types of *constraints* typically captured in object-oriented design: pre-conditions, post-conditions, and invariants.

Contracts are used primarily to establish the pre-conditions and post-conditions for a method to be able to execute properly. A *pre-condition* is a constraint that must be met for a method to execute. For example, the parameters passed to a method must be valid for the method to execute. Otherwise, an exception should be raised. A *post-condition* is a constraint that must be met after the method executes, or the effect of the method execution must be undone. For example, the method can not make any of the attributes of the object take on an invalid value. In this case, an exception should be raised, and the effect of the method's execution should be undone.

Where pre-conditions and post-conditions model the constraints on an individual method, *invariants* model constraints that must always be true for all instances of a class. Examples of invariants include domains or types of attributes, multiplicity of attributes, and the valid values of attributes. This includes the attributes that model association and aggregation relationships. For example, if an association relationship is required, an invariant should be created that will enforce it to have a valid value for the instance to exist. Invariants are normally attached to the class. As such, we can attach invariants to the CRC cards or class diagram by adding a set of assertions to them.

In Figure 10-11, the back of the CRC card constrains the attributes of an Order to specific types. For example, Order Number must be an unsigned long, and Customer must be an instance of the Customer class. Furthermore, additional invariants were added to four of the attributes. For example, Cust ID must not only be an unsigned long, but it also must have one and only one value (i.e., a multiplicity of (1..1)), and it must have the same value as the result of the GetCustID() message sent to the instance of Customer stored in the

²⁵ The idea of using contracts in design evolved from the “Design by Contract” technique developed by Bertrand Meyer. See Bertrand Meyer, *Object-Oriented Software Construction* (Englewood Cliffs, NJ: Prentice Hall, 1988).

²⁶ See Jos Warmer and Anneke Kleppe, *The Object Constraint Language: Precise Modeling with UML* (Reading, MA: Addison Wesley, 1999).

Customer attribute. Also shown is the constraint for an instance to exist, an instance of the Customer class, an instance of the State class, and at least one instance of the Product class must be associated with the Order object (see Relationships section of the CRC card). Figure 10-12 portrays the same set of constraints on a class diagram. However, if all invariants are placed on a class diagram, the diagram will become very difficult to understand. As such, we recommend extending the CRC card to document the invariants instead of attaching them all to the class diagram.

FIGURE 10-11

Invariants on a CRC Card

Elements of a Contract

Contracts document the message passing that takes place between objects. Technically speaking, a contract should be created for each message sent and received by each object; one for each interaction. However, there would be quite a bit of duplication if this were

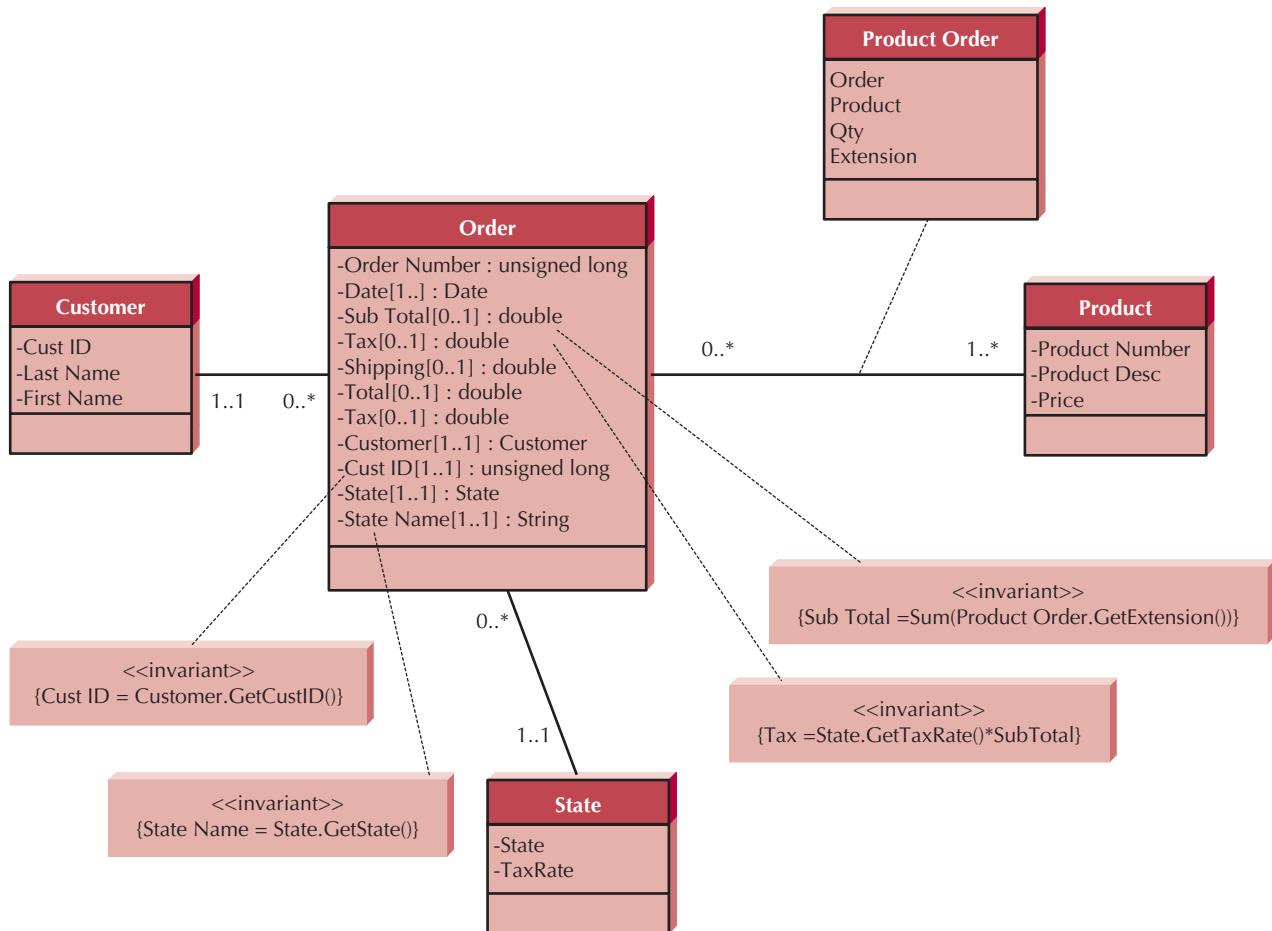


FIGURE 10-12 Invariants on a Class Diagram

YOUR TURN

10-4 Invariants

Using the CRC card in Figure 10-11 and the class diagram in Figure 10-12 as guides, add invariants for the Customer class, the State class, the Product class, and the Product-Order Association to their respective CRC cards and the class diagram.

Question:

- How easy is it to interpret the class diagram once all of the invariants were added? Look at the class

diagram in Figure 7-13 and the package diagram in Figure 9-8. What would they look like if the invariants were all attached to the diagrams? What would you recommend doing to avoid this situation?

**YOUR
TURN**

10-5 Campus Housing

In Your Turn 7-3, you created a set of CRC cards and a class diagram. Add invariants to the class diagram and to

the set of CRC cards.

done. In practice, a contract is created for each method that can receive messages from other objects (i.e., one for each visible method).

A contract should contain the information necessary for a programmer to understand what a method is to do (i.e., they are declarative in nature). This information includes the method name, class name, ID number, client objects, associated use cases, description, arguments received, type of data returned, and the pre- and post-conditions.²⁷ Contracts do not have a detailed algorithmic description of how the method is to work (i.e., they are not procedural in nature). Detailed algorithmic descriptions typically are documented in a method specification (this is described later in this chapter). In other words, a contract is composed of the information required for the developer of a client object to know what messages can be sent to the server objects and what the client can expect in return. Figure 10-13 shows a sample format for a contract.

Since each contract is associated with a specific method and a specific class, the contract must document them. The ID number of the contract is used to provide a unique identifier for every contract. The Clients (Consumers) element of a contract is a list of

Method Name:	Class Name:	ID:
Clients (Consumers):		
Associated Use Cases:		
Description of Responsibilities:		
Arguments Received:		
Type of Value Returned:		
Pre-Conditions:		
Post-Conditions:		

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 10-13
Sample Contract Format

²⁷ Currently, there is no standard format for a contract. The contract in Figure 10-13 is based on material contained in Ian Graham, *Migrating to Object Technology* (Reading, MA: Addison-Wesley, 1995), Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design* (Englewood Cliffs, NJ: Prentice-Hall, 1998); Meyer, *Object-Oriented Software Construction*, and R. Wirfs-Brock, B. Wilkerson, and L. Wiener, *Designing Object-Oriented Software* (Englewood Cliffs, NJ: Prentice Hall, 1990).

classes and methods that send a message to this specific method. This list is determined by reviewing the sequence diagrams associated with the server class. The Associated Use Cases element is a list of Use Cases in which this method is used to realize the implementation of the use case. The use cases listed here can be found by reviewing the server class's CRC card and the associated sequence diagrams.

The Description of Responsibilities provides an informal description of what the method is to perform, not how it is to do it. The arguments received are the data types of the parameters passed to the method and the value returned is the data type of the value that the method will return to its clients. Together with the method name, they form the signature of the method.

The pre-condition and post-condition element is where the pre- and post-conditions for the method are recorded. Recall that pre- and post-conditions can be written in natural language, Structured English, pseudocode, or a formal language. It really does not matter which one you use. However, given the volume of offshore outsourcing, the more precise they are written, the least likely it is that the programmer will misunderstand them. As such, we recommend that you use either pseudocode or a formal language such as UML's Object Constraint Language.²⁸

METHOD SPECIFICATION

Once the analyst has communicated the big picture of how the system needs to be put together, he or she needs to describe the individual classes and methods in enough detail so that programmers can take over and begin writing code. Methods on the CRC cards, class diagram, and contracts are described using *method specifications*. Method specifications are written documents that include explicit instructions on how to write the code to implement the method. Typically, project team members write a specification for each method and then pass them along to programmers, who write the code during the Implementation phase of the project. Specifications need to be very clear and easy to understand, or programmers will be slowed down trying to decipher vague or incomplete instructions.

There is no formal syntax for a method specification, so every organization uses its own format, often using a form like the one in Figure 10-14. Typical method specification forms contain four components that convey the information that programmers will need to write the appropriate code: general information, events, message passing, and algorithm specification.

General Information

The top of the form in Figure 10-14 contains general information, such as the name of the method, name of the class in which this implementation of the method will reside, ID number, Contract ID that identifies the contract associated with this method implementation, programmer assigned, the date due, and the target programming language. This information is used to help manage the programming effort.

**YOUR
TURN**

10-6 Contract

Using the CRC card in Figure 10-11, the class diagram in Figure 10-12, and the sample contract format in Figure 10-13

as guides, create contracts for the Calculate subtotal, Calculate tax, Calculate shipping, and Calculate total methods.

²⁸ See Warmer and Kleppe, *The Object Constraint Language: Precise Modeling with UML*.

Method Name:	Class Name:	ID:
Contract ID:	Programmer:	Date Due:
Programming Language: <input type="checkbox"/> Visual Basic <input type="checkbox"/> Smalltalk <input type="checkbox"/> C++ <input type="checkbox"/> Java		
Triggers/Events:		
Arguments Received:	Notes:	
Data Type:		
Messages Sent & Arguments Passed:	Data Type:	Notes:
ClassName.MethodName:		
Argument Returned:	Notes:	
Data Type:		
Algorithm Specification:		
Misc.Notes:		

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 10-14
Method Specification
Form

Events

The second section of the form is used to list the events that trigger the method. An *event* is a thing that happens or takes place. Clicking the mouse generates a mouse event, pressing a key generates a keystroke event—in fact, almost everything the user does generates an event to occur.

In the past, programmers used procedural programming languages (e.g., COBOL, C) that contained instructions that were implemented in a predefined order, as determined by the computer system, and users were not allowed to deviate from the order. Many programs today are *event-driven* (e.g., programs written in languages such as Visual Basic, Smalltalk, C++, or Java), and event-driven programs include methods that are executed in response to an event initiated by the user, system, or another method. After initialization, the system waits for an event to occur. When it does, a method is fired which carries out the appropriate task, and then the system waits once again.

We have found that many programmers still use method specifications when programming in event-driven languages, and they include the event section on the form to capture when the method will be invoked. Other programmers have switched to other design tools that capture event-driven programming instructions, such as the behavioral state machine described in Chapter 8.

Message Passing

The next set of sections of the method specification describes the message passing to and from the method, which are identified on the sequence and collaboration diagrams. Programmers will need to understand what arguments are being passed into, from, and returned by the method since the arguments ultimately will translate into attributes and data structures within the actual method.

Algorithm Specification

Algorithm specifications can be written in Structured English, some form of pseudocode, or some type of formal language.²⁹ *Structured English* is simply a formal way of writing instructions that describe the steps of a process. Because it is the first step toward the implementation of the method, it looks much like a simple programming language. Structured English uses short sentences that clearly describe exactly what work is performed on what data. There are many versions of Structured English because there are no formal standards; each organization has its own type of Structured English.

Figure 10-15 shows some examples of commonly used Structured English statements. Action statements are simple statements that perform some action. An If statement

Common Statements	Example
Action Statement	Profits = Revenues – Expenses Generate Inventory-Report
If Statement	IF Customer Not in the Customer Object Store THEN Add Customer record to Customer Object Store ELSE Add Current-Sale to Customer's Total-Sales Update Customer record in Customer Object Store
For Statement	FOR all Customers in Customer Object Store DO Generate a new line in the Customer-Report Add Customer's Total-Sales to Report-Total
Case Statement	CASE IF Income < 10,000: Marginal-tax-rate = 10 percent IF Income < 20,000: Marginal-tax-rate = 20 percent IF Income < 30,000: Marginal-tax-rate = 31 percent IF Income < 40,000: Marginal-tax-rate = 35 percent ELSE Marginal-Tax-Rate = 38 percent ENDCASE

FIGURE 10-15
Structured English

²⁹ For our purposes, Structured English or pseudocode will suffice. However, there has been some work with the Catalysis, Fusion, and Syntropy methodologies to include formal languages, such as VDM and Z, into specifying object-oriented systems.

controls actions that are performed under different conditions, and a For statement (or a While statement) performs some actions until some condition is reached. Finally, a Case statement is an advanced form of an If statement that has several mutually exclusive branches.

Pseudocode is a language that contains logical structures including sequential statements, conditional statements, and iteration. It differs from Structured English in that pseudocode contains details that are programming-specific, like initialization instructions or linking, and it also is more extensive so that a programmer can write the module by mirroring the pseudocode instructions. In general, pseudocode is much more like real code, and its audience is the programmer as opposed to the analyst. Its format is not as important as the information it conveys. Figure 10-16 shows a short example of pseudocode for a module that is responsible for getting CD information.

Writing good pseudocode can be difficult—imagine creating instructions that someone else can follow without having to ask for clarification or make a wrong assumption. For example, have you ever given a friend directions to your house, and they ended up getting lost? To you, the directions may have been very clear, but that is because of your personal assumptions. To you, the instruction “take the first left turn” may really mean, “take a left turn at the first stoplight.” Someone else may have made the interpretation of “take a left turn at the first road, with or without a light.” Therefore, when writing pseudocode, pay special attention to detail and readability.

If the algorithm of a method is complex, a tool that can be useful for algorithm specification is UML’s *activity diagram* (see Chapter 6). If you recall, activity diagrams can be used to specify any type of process. Obviously, an algorithm specification represents a process. However, due to the nature of object-orientation, processes tend to be highly distributed over many little methods over many objects. As such, needing to use an activity diagram to specify the algorithm of a method may, in fact, hint at a problem in the design. For example, the method should be further decomposed or there could be missing classes.

The last section of the method specification provides space for other information that needs to be communicated to the programmer, such as calculations, special business rules, calls to subroutines or libraries, and other relevant issues. This also can point out changes or improvements that will be made to any of the other design documentation based on problems that the analyst detected during the specification process.³⁰

FIGURE 10-16
Pseudocode

```
(Get_CD_Info module)
Accept (CD.Title) {Required}
Accept (CD.Artist) {Required}
Accept (CD.Category) {Required}
Accept (CD.Length)
Return
```

YOUR TURN

10-7 Method Specification

Using the CRC card in Figure 10-11, the class diagram in Figure 10-12, and the contract created in Your Turn 10-6 as guides, create method specifications for the Calculate

subtotal, Calculate tax, Calculate shipping, and Calculate total methods.

³⁰ Remember that the development process is very incremental and iterative in nature. Therefore, changes could be cascaded back to any point in the development process (e.g., to use case descriptions, use case diagrams, CRC cards, class diagrams, object diagrams, sequence diagrams, communication diagrams, behavioral state machines, and package diagrams).

APPLYING THE CONCEPTS AT CD SELECTIONS

Alec Adams, the senior systems analyst and project manager for the Internet sales system, and his team began the detailed object design process by reviewing the class and package diagram for the problem domain layer (see Figures 7-13 and 9-8). Upon their review, it was discovered that there were quite a few many-to-many (*..*) association relationships on the class diagram. Alec questioned whether this was a correct representation of the actual situation. Brian, one of the staff members on the Internet sales system project, admitted that when they put together the class diagram, they had decided to model most of the associations as a many-to-many multiplicity, figuring that this could be easily fixed at a later point in time when they had more precise information. Since Brian was the team member that was most familiar with structural modeling and was the analyst in charge of the data management layer (see Chapter 11), Alec assigned him to evaluate the multiplicity of each association in the model and to restructure and optimize the evolving problem domain model.

Figure 10-17 shows the updated version of the class diagram. As you can see, Brian included both the lower and upper values of the multiplicity of the associations. He did this to remove any ambiguity about the associations. Even though there is a one-to-one relationship between the CD class and the Mkt Info class, Brian considered merging them into a single class. However, he decided that they were sufficiently different to keep them separate. He reasoned this by assuming that not all customers would want to see all of the Mkt Info associated with every CD.

Upon reviewing the new revised class diagram, Alec assigned different members of his team to the different packages identified (see Figure 9-8). Because Brian had already spent quite a bit of time on the classes in the CD package, Alec assigned it to him. The classes in the CD package were CD, Vendor, Mkt Info, Review, Artist Info, and Sample Clip.

Next, Brian added invariants, pre-conditions, and post-conditions to the classes and their methods. For example, Figure 10-18 portrays the back of the CRC card for the CD class. He decided to add only the invariant information to the CRC cards and not the class diagram to keep the class diagram as simple and as easy to understand as possible. Notice the additional set of multiplicity, domain, and referential integrity invariants added to the attributes and relationships. During this process, he realized that not all CDs had marketing information associated with them. As such, he modified the multiplicity from the CD class to the Mkt Info class. Furthermore, he needed to add contracts for each method. Figure 10-19 portrays the contract for the GetReview() method associated with the Mkt Info class. Notice that there is a pre-condition for this method to succeed—Review attribute not Null.

Upon completing the CRC cards and contracts, Brian moved on to specifying the detailed design for each method. For example, the method specification for the GetReview() method is given in Figure 10-20. Brian developed this specification by reviewing the Places Order use case (see Figure 6-15), the sequence diagram (see Figure 8-5), and the contract (see Figure 10-19). Notice that Brian is enforcing the pre-condition on the contract by testing to see whether the Review attribute that contains the list of reviews contains a value or not. Since the method is to be implemented in Java, he has specified that an exception is to be thrown if there are no reviews.

Finally, Brian updated the class diagram for the CD package (see Figure 10-21).

SUMMARY

Revisiting the Basic Characteristics of Object-Oriented Systems

A class is a template on which objects can be instantiated. An object is a person, place, or thing about which we want to capture information. Each object has attributes and

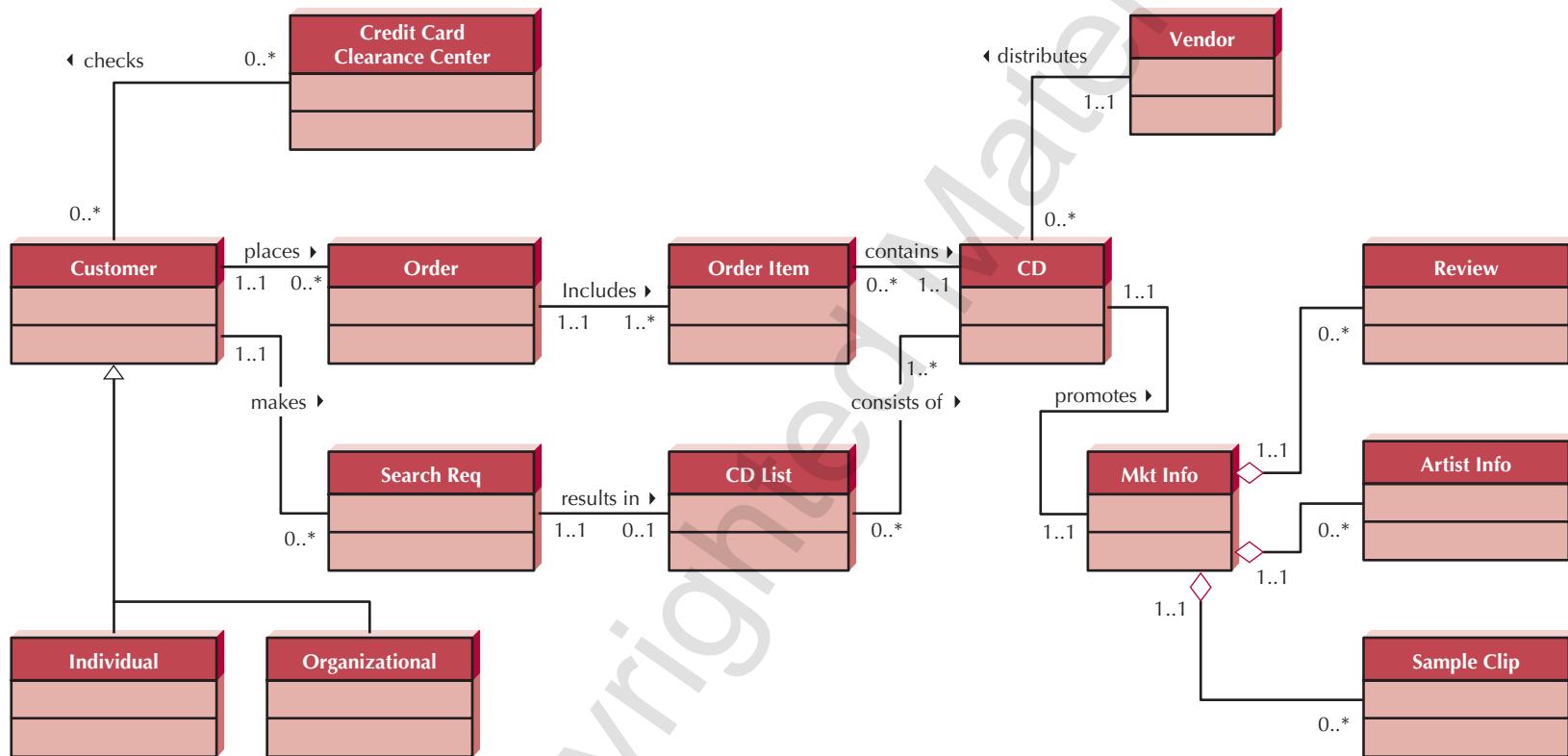


FIGURE 10-17 Revised CD Selections Internet Sales System Class Diagram (Places Order Use Case View)

Back:

Attributes:	
CD Number (1..1) (unsigned long)	
CD Name (1..1) (String)	
Pub Date (1..1) (Date)	
Artist Name (1..1) (String)	
Artist Number (1..1) (unsigned long)	
Vendor (1..1) (Vendor)	
Vendor ID (1..1) (unsigned long) {Vendor ID = Vendor.GetVendorID()}	
Relationships:	
Generalization (a-kind-of):	
Aggregation (has-parts):	
Other Associations:	Order Item {0..*} CD List {0..*} Vendor {1..1} Mkt Info {0..1}

FIGURE 10-18
Back of CD CRC Card

Method Name:	GetReview()	Class Name:	ID:
Clients (Consumers):	CD Detailed Report		
Associated Use Cases:	Places Order		
Description of Responsibilities:	Return review objects for the Detailed Report Screen to display		
Arguments Received:			
Type of Value Returned:	List of Review objects		
Pre-Conditions:	Review attribute not Null		
Post-Conditions:			

FIGURE 10-19
Get Review Method
Contract

methods. The methods are executed by objects sending messages that trigger them. Encapsulation and information hiding allows an object to conceal its inner processes and data from the other objects. Polymorphism and dynamic binding allows a message to be interpreted differently by different kinds of objects. However, if polymorphism is not used in a semantically consistent manner, polymorphism can make an object design incomprehensible. Classes can be arranged in a hierarchical fashion in which subclasses inherit attributes and methods from superclasses to reduce the redundancy in development. However, through redefinition capabilities or multiple inheritance, inheritance conflicts can be introduced into the design.

Triggers/Events: Detail Button on Basic Report is pressed		
Arguments Received: Data Type:		Notes:
Messages Sent & Arguments Passed: ClassName.MethodName:	Data Type:	Notes:
Arguments Returned: Data Type: List		Notes: List of Review objects
Algorithm Speculation: <pre>IF Review Not Null Return Review Else Throw Null Exception</pre>		
Misc. Notes:		

FIGURE 10-20
Create Review Method Specification

Design Criteria

Coupling, cohesion, and connascence have been put forth as a set of criteria by which a design of an object-oriented system can be evaluated. Two types of coupling, interaction and inheritance, and three types of cohesion, method, class, and generalization/specialization, were described. Interaction coupling deals with the communication that takes place among the objects, and inheritance coupling deals with the innate dependencies in the use of inheritance in object-oriented systems. Method cohesion addresses how single-minded a method is. The fewer things a method does, the more cohesive it is. Class cohesion does the same for classes. A class should be a representation of one and only one thing. Generalization/specialization cohesion deals with the quality of an inheritance hierarchy. Good inheritance hierarchies only support generalization and specialization (A-Kind-Of) semantics and the principle of substitutability. Connascence generalizes coupling and cohesion and then combines them with the different levels of encapsulation. The general rule of thumb is to maximize the cohesion (connascence) within an encapsulation boundary and minimize the coupling (connascence) between the encapsulation boundaries.

Object Design Activities

There are five basic object design activities. First, additional specification is possible by carefully reviewing the models, deciding on the proper visibility of the attributes and methods, setting the signature for each method, and identifying any constraints associated with the classes or the classes' methods. Second, look for opportunities for reuse by reviewing the model and looking at possible patterns, class libraries, frameworks, and components that could be used to enhance the system. Third, restructure the model through the use of factoring and normalization. Be sure to take the programming language into consideration. It may be necessary to map the current design into the restricted capabilities of the language

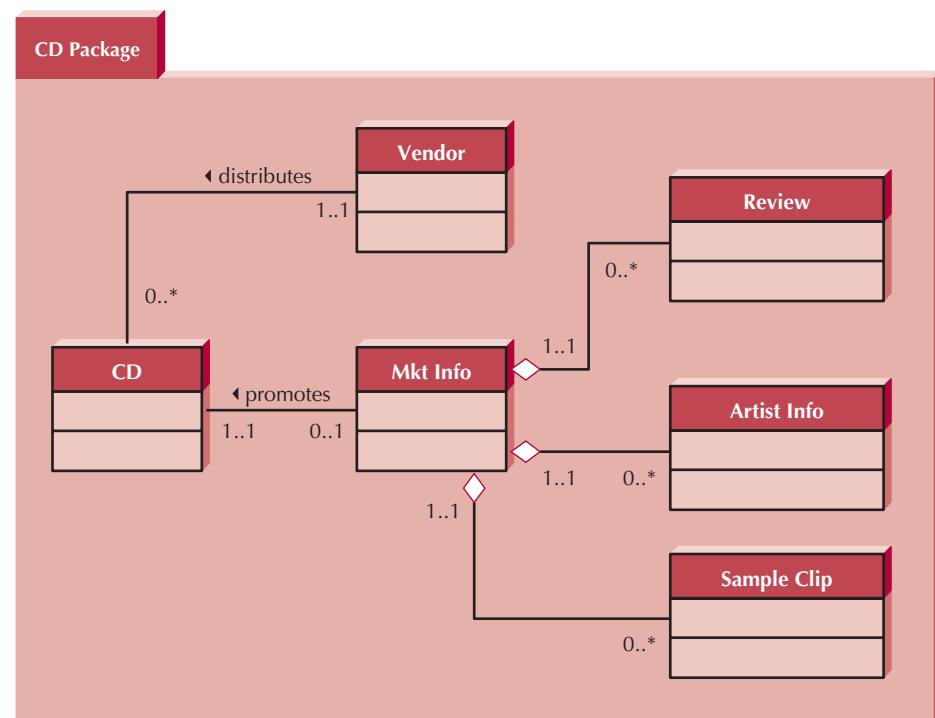


FIGURE 10-21
Revised Package Diagram for the CD Package on the PD Layer of CD Selections Internet Sales System

(e.g., the language only supports single inheritance). Also, be certain that the inheritance in your model only supports generalization/specialization (A-Kind-Of) semantics and the principle of substitutability. Fourth, optimize the design. However, be careful in the process of optimizing the design. Optimizations typically decrease the understandability of the model. Fifth, map the problem domain classes to an implementation language.

Constraints and Contracts

There are three types of constraints associated with object-oriented design: invariants, pre-conditions, and post-conditions. Invariants capture constraints that must always be true for all instances of a class (e.g., domains and values of attributes or multiplicity of relationships). Typically, invariants are attached to class diagrams and CRC cards. However, for clarity purposes, we suggest only placing them on the CRC cards.

Contracts formalize the interaction between objects (i.e., the message passing). As such, they include the pre- and post-conditions that must be enforced for a method to execute properly. Contracts provide an approach to model the rights and obligations when client and server objects interact. From a practical perspective, every interaction that can take place between all possible client and server objects are not modeled on separate contracts. Instead, a single contract is drawn up for using each visible method of a server object.

Method Specification

A method specification is a written document that provides clear and explicit instructions on how a method is to behave. Without clear and unambiguous method specifications, critical design decisions will have to be made by programmers instead of by designers. Given the volume of offshore outsourcing, this is not advisable. Even though there is no standard format for a method specification, typically four types of information are captured. First, there is general information such as the name of the method, name of the class, Contract ID, programmer assigned, the date due, and the target programming language. Second, due to the rise in popularity of GUI-based and event-driven systems, events also are captured. Third, the information that deals with the signature of the method, data received, data passed on to other methods, and data returned by the method is recorded. Finally, an unambiguous specification of the algorithm is given. The algorithm typically is modeled using Structured English, pseudocode, an activity diagram, or a formal language.

KEY TERMS

Active value	Contract	Ideal class cohesion
ActiveX	Coupling	Information hiding
Activity diagram	Derived attribute	Inheritance
Application program interface (API)	Design Patterns	Inheritance conflict
Attribute	Dynamic binding	Inheritance coupling
Class	Encapsulation	Instance
Class cohesion	Event	Interaction coupling
Class library	Event-driven	Invariant
Client	Exceptions	JavaBean
Cohesion	Factoring	Law of Demeter
Component	Fan-out	Message
Connascence	Framework	Method
Constraint	Generalization/Specialization cohesion	Method cohesion
Consumer	Homonyms	Method specification

Multiple inheritance	Pre-condition	Structured English
Normalization	Pseudocode	Substitutability
Object	Redefinition	Supplier
Object-based language	Server	Synonyms
Patterns	Signature	Trigger
Polymorphism	Single inheritance	Visibility
Post-condition	State	

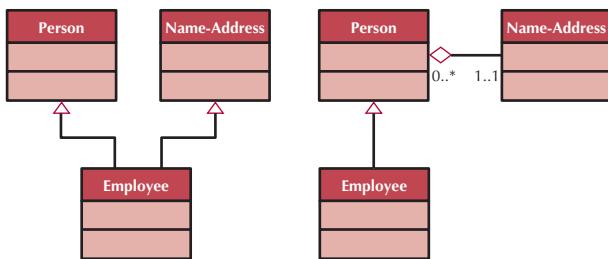
QUESTIONS

1. What are the basic characteristics of object-oriented systems?
2. What is dynamic binding?
3. Define polymorphism. Give one example of a “good” use of polymorphism and one example of a “bad” use of polymorphism.
4. What is an inheritance conflict? How does an inheritance conflict affect the design?
5. Why is cancellation of methods a bad thing?
6. Give the guidelines to avoid problems with inheritance conflicts.
7. How important is it to know which object-oriented programming language is going to be used to implement the system?
8. What are the additional types of inheritance conflicts are there when using multiple inheritance?
9. What is the Law of Demeter?
10. What are the six types of interaction coupling? Give one example of “good” interaction coupling and one example of “bad” interaction coupling.
11. What are the seven types of method cohesion? Give one example of “good” method cohesion and one example of “bad” method cohesion.
12. What are the four types of class cohesion? Give one example of each type.
13. What are the five types of connascence described in your text? Give one example of each type.
14. When designing a specific class, what types of additional specification for a class could be necessary?
15. What are constraints? What are the three different types of constraints?
16. What are exceptions?
17. What are patterns, frameworks, class libraries, and components? How are they used to enhance the evolving design of the system?
18. How are factoring and normalization used in designing an object system?
19. What are the different ways to optimize an object system?
20. What is the typical downside of system optimization?
21. What is the purpose of a contract? How are contracts used?
22. What is an invariant? How are invariants modeled in a design of a class? Give an example of an invariant for a hourly employee class.
23. Create a contract for a Compute-Pay method associated with an hourly employee class.
24. How do you specify a method’s algorithm? Give an example of an algorithm specification for a Compute-Pay method associated with an hourly employee class.
25. How are methods specified? Give an example of a method specification for a Compute-Pay method associated with an hourly employee class.

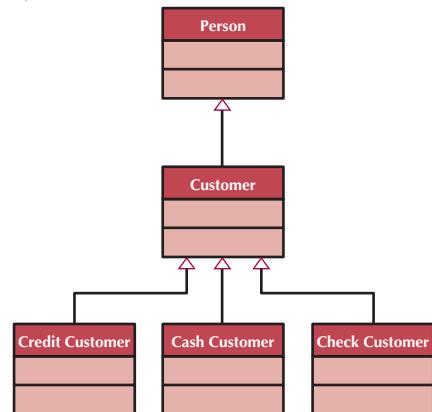
EXERCISES

- A. For the Health Club exercises in Chapter 6 (N, O), Chapter 7 (O), Chapter 8 (C, D), and Chapter 9 (D), choose one of the classes and create a set of invariants for attributes and relationships and add them to the CRC card for the class.
- B. Choose one of the methods in the class that you chose for Exercise A and create a contract and a method specification for it. Use Structured English for the algorithm specification.
- C. For the Picnics R Us exercises in Chapter 6 (P, Q), Chapter 7 (P), Chapter 8 (H), and Chapter 9 (E), choose one of the classes and create a set of invariants for attributes and relationships and add them to the CRC card for the class.
- D. Choose one of the methods in the class that you chose for Exercise C and create a contract and a method specification for it. Use Structured English for the algorithm specification.

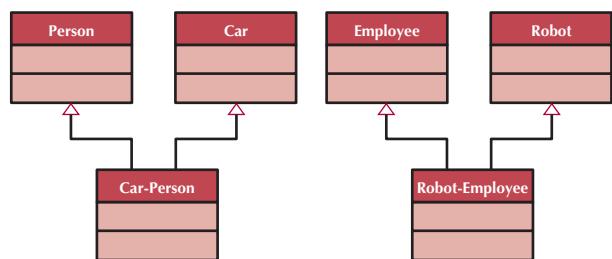
- E. For the Of-the-Month Club exercises in Chapter 6 (R, S), Chapter 7 (Q), Chapter 8 (I), and Chapter 9 (F), choose one of the classes and create a set of invariants for attributes and relationships and add them to the CRC card for the class.
- F. Choose one of the methods in the class that you chose for Exercise E and create a contract and a method specification for it. Use Structured English for the algorithm specification.
- G. For the Library exercises in Chapter 6 (T, U), Chapter 7 (R), Chapter 8 (J), and Chapter 9 (G), choose one of the classes and create a set of invariants for attributes and relationships and add them to the CRC card for the class.
- H. Choose one of the methods in the class that you chose for Exercise G and create a contract and a method specification for it. Use Structured English for the algorithm specification.
- I. Describe the difference in meaning between the following two class diagrams. Which is a better model? Why?



- J. From a cohesion, coupling, and connascence perspective, is the following class diagram a good model? Why or why not?



- K. From a cohesion, coupling, and connascence perspective, are the following class diagrams good models? Why or why not?



- L. Create a set of inheritance conflicts for the two inheritance structures in the class diagrams of Exercise K.

MINICASES

1. Your boss has been in the software development field for thirty years. He has always prided himself on his ability to adapt his skills from one approach to developing software to the next approach. For example, he had no problem learning structured analysis and design in the early 1980s and information engineering in the early 1990s. He even understands the advantage of rapid application development. But the other day, when you and he were talking about the advantages of object-oriented approaches, he became totally confused. He thought that characteristics such as polymorphism and inheritance were an advantage for object-oriented systems. However, when you explained the problems with inheritance conflicts, redefinition capabilities, and the need for semantic consistency across different implementations of

methods, he was ready to simply give up. To make matters worse, you then went on to explain the importance of contracts in controlling the development of the system. At this point in the conservation, he basically threw in the towel. As he walked off, you heard him say something like “I guess it’s true, it’s too hard to teach an old dog new tricks.”

Being a loyal employee and friend, you decided to write a short tutorial to give your boss on object-oriented systems development. As a first step, create a detailed outline for the tutorial. As a subtle example, use good design criteria, such as coupling and cohesion, in the design of your tutorial outline.

2. You have been working with the Holiday Travel Vehicle problem for quite a while. You should go back and refresh your memory about the problem before

attempting to solve this situation. Refer back to your solutions for Minicase 2 in Chapter 7 and Minicase 2 in Chapter 8.

In the new system for Holiday Travel Vehicles, the system users follow a two-stage process to record complete information on all of the vehicles sold. When an RV or trailer first arrives at the company from the manufacturer, a clerk from the inventory department creates a new vehicle record for it in the computer system. The data entered at this time include basic descriptive information on the vehicle such as manufacturer, name, model, year, base cost, and freight charges. When the vehicle is sold, the new vehicle record is updated to reflect the final sales terms and the dealer-installed options added to the vehicle. This information is entered into the system at the time of sale when the salesperson completes the sales invoice.

When it is time for the clerk to finalize the new vehicle record, the clerk will select a menu option from the system, which is called Finalize New Vehicle Record. The tasks involved in this process are described below.

When the user selects the “Finalize New Vehicle Record” from the system menu, the user is immediately prompted for the serial number of the new

vehicle. This serial number is used to retrieve the new vehicle record for the vehicle from system storage. If a record cannot be found, the serial number is probably invalid. The vehicle serial number is then used to retrieve the option records that describe the dealer-installed options that were added to the vehicle at the customer’s request. There may be zero or more options. The cost of the option specified on the option record(s) is totaled. Then, the dealer cost is calculated using the vehicle’s base cost, freight charge, and total option cost. The completed new vehicle record is passed back to the calling module.

- a. Update the structural model (CRC cards and class diagram) with this additional information.
- b. For each class in the structural model create a set of invariants for attributes and relationships and add them to the CRC cards for the classes.
- c. Choose one of the classes in the structural model. Create a contract for each method in that class. Be as complete as possible.
- d. Create a method specification for each method in the class you chose for question c. Use Structured English for the algorithm specification.

CHAPTER 11

DATA MANAGEMENT LAYER DESIGN

A project team designs the data management layer of the system using a four-step process: selecting the format of the storage, mapping the problem domain classes to the selected format, optimizing the storage to perform efficiently, and then designing the necessary data access and manipulation classes. This chapter first describes the different ways in which objects can be stored and several important characteristics that should be considered when choosing among object-persistence formats. Second, the chapter describes a problem domain class to object-persistence format mapping process for the most important object-persistence formats. Third, since the most popular storage format today is the relational database, the chapter focuses on the optimization of relational databases from both storage and access perspectives. Last, the chapter describes how to design data access and manipulation classes.

OBJECTIVES

- Become familiar with several object-persistence formats.
- Be able to map problem domain objects to different object-persistence formats.
- Be able to apply the steps of normalization to a relational database.
- Be able to optimize a relational database for object storage and access.
- Become familiar with indexes for relational databases.
- Be able to estimate the size of a relational database.
- Be able to design the data access and manipulation classes.

CHAPTER OUTLINE

Introduction	Optimizing RDBMS-Based Object Storage
Object-Persistence Formats	Optimizing Storage Efficiency
Sequential and Random Access Files	Optimizing Data Access Speed
Relational Databases	Estimating Data Storage Size
Object-Relational Databases	Designing Data Access and Manipulation Classes
Object-Oriented Databases	Applying the Concepts at CD Selections
Selecting an Object-Persistence Format	Select Object-Persistence Format
Mapping Problem Domain Objects to Object-Persistence Formats	Mapping Problem Domain Objects to the Object-Persistence Format
Mapping Problem Domain Objects to an OODBMS Format	Data Optimization and Sizing
Mapping Problem Domain Objects to an ORDBMS Format	Data Access and Manipulation
Mapping Problem Domain Objects to an RDBMS Format	Class Design
	Summary

INTRODUCTION

As explained in Chapter 9, the work done by any application can be divided into a set of layers. This chapter focuses on the *data management layer*, which includes both data access and manipulation logic along with the actual design of the storage. The data storage component of the data management layer manages how data is stored and handled by the programs that run the system. This chapter describes how a project team designs the storage for objects (object persistence) using a four step approach: selecting the format of the storage, mapping the problem domain objects to the object-persistence format, optimizing the object-persistence format, and designing the data access and manipulation classes necessary to handle the communication between the system and the database.

Applications are of little use without the data that they support. How useful is a multimedia application that can't support images or sound? Why would someone log into a system to find information if it took him or her less time to locate the information manually? The design phase of the system development life cycle (SDLC) includes four steps to object-persistence design that decrease the chances of ending up with inefficient systems, long system response times, and users who cannot get to the information that they need in the way that they need it—all of which can affect the success of the project.

The first part of this chapter describes a variety of storage formats and explains how to select the appropriate one for your application. From a practical perspective, there are four basic types of formats that can be used to store objects for application systems: files (sequential and random), object-oriented databases, object-relational databases, or relational databases.¹ Each type has certain characteristics that make it more appropriate for some types of systems over others.

Once the object-persistence format is selected to support the system, the problem domain objects need to drive the design of the actual object storage. Then the object storage needs to be designed to optimize its processing efficiency, which is the focus of the next part of the chapter. One of the leading complaints by end users is that the final system is *too slow*, so to avoid such complaints project team members must allow time during the design phase to carefully make sure that the file or database performs as fast as possible. At the same time, the team must keep hardware costs down by minimizing the storage space that the application will require. The goals to maximize access to the objects and minimize the amount of space taken to store objects can conflict, and designing object persistence efficiency usually requires trade-offs.

Finally, it is necessary to design a set of data access and manipulation classes to ensure the independence of the problem domain classes from the storage format. The data access and manipulation classes handle all communication with the database. In this manner, the problem domain is decoupled from the object storage allowing the object storage to be changed without impacting the problem domain classes.

OBJECT-PERSISTENCE FORMATS

There are four main types of *object-persistence formats*: files (sequential and random access), object-oriented databases, object-relational databases, and relational databases. *Files* are electronic lists of data that have been optimized to perform a particular transaction. For example, Figure 11-1 shows a customer order file with information about customers' orders, in the form in which it is used, so that the information can be accessed and processed quickly by the system.

¹ There are other types of files, such as relative, indexed sequential, and multi-indexed sequential, and databases, such as hierarchical, network, and multidimensional. However, these formats typically are not used for object persistence.

Order Number	Date	Cust ID	Last Name	First Name	Amount	Tax	Total	Prior Customer	Payment Type
234	11/23/00	2242	DeBerry	Ann	\$ 90.00	\$ 5.85	\$ 95.85	Y	MC
235	11/23/00	9500	Chin	April	\$ 12.00	\$ 0.60	\$ 12.60	Y	VISA
236	11/23/00	1556	Fracken	Chris	\$ 50.00	\$ 2.50	\$ 52.50	N	VISA
237	11/23/00	2242	DeBerry	Ann	\$ 75.00	\$ 4.88	\$ 79.88	Y	AMEX
238	11/23/00	2242	DeBerry	Ann	\$ 60.00	\$ 3.90	\$ 63.90	Y	MC
239	11/23/00	1035	Black	John	\$ 90.00	\$ 4.50	\$ 94.50	Y	AMEX
240	11/23/00	9501	Kaplan	Bruce	\$ 50.00	\$ 2.50	\$ 52.50	N	VISA
241	11/23/00	1123	Williams	Mary	\$ 120.00	\$ 9.60	\$ 129.60	N	MC
242	11/24/00	9500	Chin	April	\$ 60.00	\$ 3.00	\$ 63.00	Y	VISA
243	11/24/00	4254	Bailey	Ryan	\$ 90.00	\$ 4.50	\$ 94.50	Y	VISA
244	11/24/00	9500	Chin	April	\$ 24.00	\$ 1.20	\$ 25.20	Y	VISA
245	11/24/00	2242	DeBerry	Ann	\$ 12.00	\$ 0.78	\$ 12.78	Y	AMEX
246	11/24/00	4254	Bailey	Ryan	\$ 20.00	\$ 1.00	\$ 21.00	Y	MC
247	11/24/00	2241	Jones	Chris	\$ 50.00	\$ 2.50	\$ 52.50	N	VISA
248	11/24/00	4254	Bailey	Ryan	\$ 12.00	\$ 0.60	\$ 12.60	Y	AMEX
249	11/24/00	5927	Lee	Diane	\$ 50.00	\$ 2.50	\$ 52.50	N	AMEX
250	11/24/00	2242	DeBerry	Ann	\$ 12.00	\$ 0.78	\$ 12.78	Y	MC
251	11/24/00	9500	Chin	April	\$ 15.00	\$ 0.75	\$ 15.75	Y	MC
252	11/24/00	2242	DeBerry	Ann	\$ 132.00	\$ 8.58	\$ 140.58	Y	MC
253	11/24/00	2242	DeBerry	Ann	\$ 72.00	\$ 4.68	\$ 76.68	Y	AMEX

FIGURE 11-1 Customer Order File

A *database* is a collection of groupings of information that is related to each other in some way (e.g., through common fields). Logical groupings of information could include such categories as customer data, information about an order, product information, and so on. A *database management system (DBMS)* is software that creates and manipulates these databases (see Figure 11-2 for a relational database example). Such *end-user DBMSs* as Microsoft Access support small-scale databases that are used to enhance personal productivity, while *enterprise DBMSs*, such as DB2, Jasmine, and Oracle can manage huge volumes of data and support applications that run an entire company. An end-user DBMS is significantly less expensive and easier for novice users to use than its enterprise counterpart, but it does not have the features or capabilities that are necessary to support mission-critical or large-scale systems.

The next set of sections describe sequential and random access files, relational databases, object-relational databases, and object-oriented databases that can be used to handle a system's object persistence requirements. Finally, we describe a set of characteristics on which the different formats can be compared.

Sequential and Random Access Files

From a practical perspective, most object-oriented programming languages support sequential and random access files as part of the language.² In this section, we describe what sequential access and random access files are.³ We also describe how sequential access and random access files are used to support an application. For example, they can be used to support master files, look-up files, transaction files, audit files, and history files.

² For example, see the FileInputStream, FileOutputStream, and RandomAccessFile classes in the java.io package.

³ For a more complete coverage of issues related to the design of files, see Owen Hanson, *Design of Computer Data Files* (Rockville, MD: Computer Science Press, 1982).

Sequential access files only allow sequential file operations to be performed (e.g., read, write, and search). Sequential access files are very efficient for sequential operations, such as report writing. However, for random operations, such as finding or updating a specific object, they are very inefficient. On the average, 50 percent of the contents of a sequential access file will have to be searched through before finding the specific object of interest in the file. They come in two flavors: ordered and unordered.

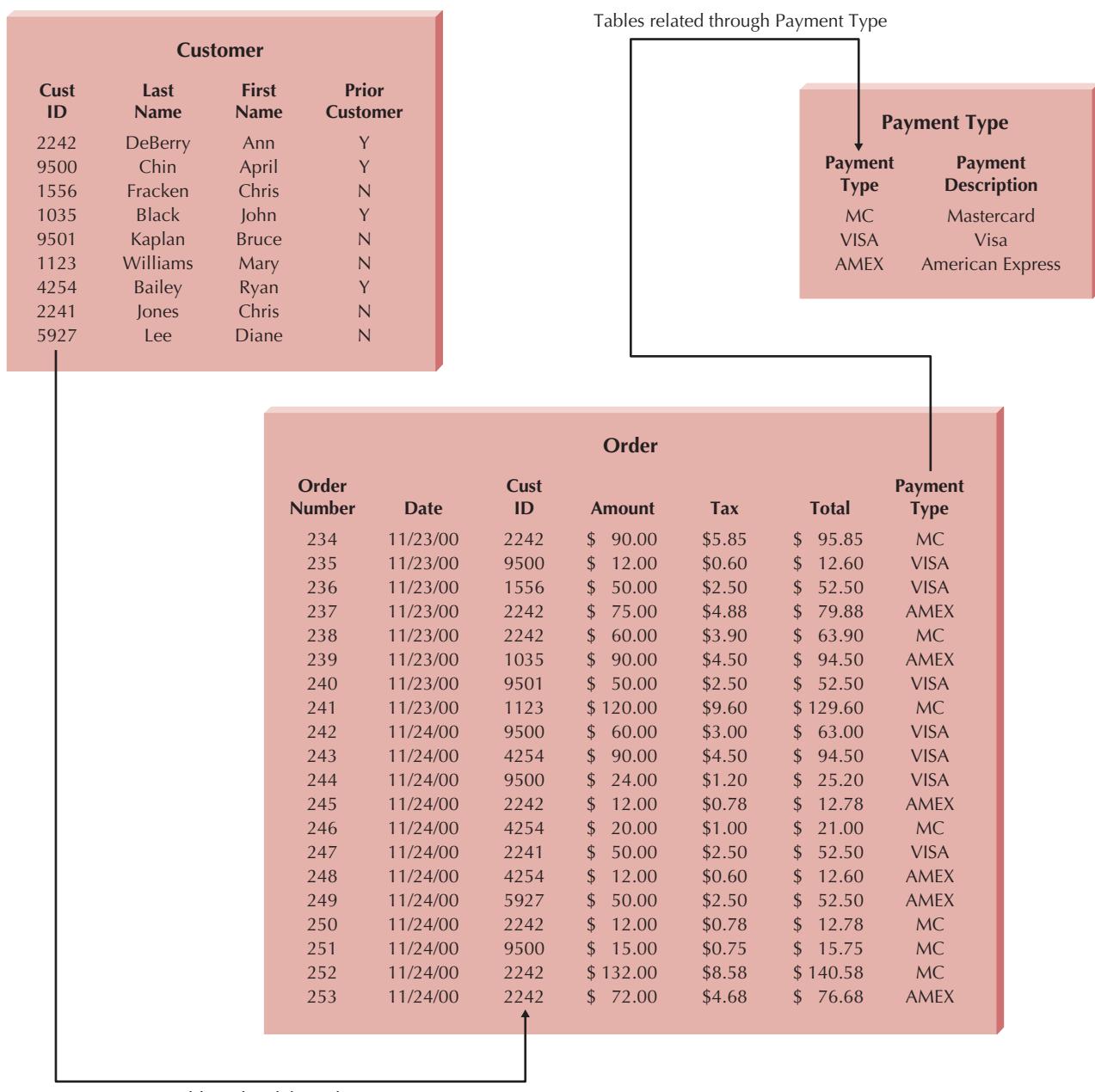


FIGURE 11-2 Customer Order Database

An *unordered sequential access file* basically is an electronic list of information that is stored on disk. Unordered files are organized serially (i.e., the order of the file is the order in which the objects are written to the file). Typically, new objects simply are added to the file's end.

Ordered sequential access files are placed into a specific sorted order (e.g., in ascending order by customer number). However, there is overhead associated with keeping files in a particular sorted order. The file designer can keep the file in sorted order by always creating a new file each time a delete or addition occurs, or he or she can keep track of the sorted order via the use of *pointers*, which is information about the location of the related record. A pointer is placed at the end of each record, and it "points" to the next record in a series or set. The underlying data/file structure in this case is a *linked list*.⁴

Random access files allow only random or direct file operations to be performed. This type of file is optimized for random operations, such as finding and updating a specific object. Random access files typically give a faster response time to find and update operations than any other type of file. However, since they do not support sequential processing, applications such as report writing are very inefficient. The various methods to implement random access files are beyond the scope of this book.⁵

There are times when it is necessary to be able to process files in both a sequential and random manner. One simple way to do this is to use a sequential file that contains a list of the keys (the field in which the file is to be kept in sorted order) and a random access file for the actual objects. This will minimize the cost of additions and deletions to a sequential file, while allowing the random file to be processed sequentially by simply passing the key to the random file to retrieve each object in sequential order. It also allows for fast random processing to occur by using only the random access file, thus optimizing the overall cost of file processing. However, if a file of objects needs to be processed in both a random and sequential manner, the developer should look toward using a database (relational, object-relational, or object-oriented) instead.

There are many different types of application files—for example, master files, look-up files, transaction files, audit files, and history files. *Master files* store core information that is important to the business and, more specifically, to the application, such as order information or customer mailing information. They usually are kept for long periods of time, and new records are appended to the end of the file as new orders or new customers are captured by the system. If changes need to be made to existing records, programs must be written to update the old information.

Look-up files contain static values, such as a list of valid zip codes or the names of the U.S. states. Typically, the list is used for validation. For example, if a customer's mailing address is entered into a master file, the state name is validated against a look-up file that contains U.S. states to make sure that the operator entered the value correctly.

A *transaction file* holds information that can be used to update a master file. The transaction file can be destroyed after changes are added, or the file may be saved in case the transactions need to be accessed again in the future. Customer address changes, for one, would be stored in a transaction file until a program is run that updates the customer address master file with the new information.

For control purposes, a company might need to store information about how data changes over time. For example, as human resource clerks change employee salaries in a human resource system, the system should record the person who made the changes to the salary amount, the date, and the actual change that was made. An *audit file* records "before"

⁴ For more information on various data structures, see Ellis Horowitz and Sartaj Sahni, *Fundamentals of Data Structures*. (Rockville, MD: Computer Science Press, 1982), and Michael T. Goodrich and Roberto Tamassia, *Data Structures and Algorithms in Java* (New York: Wiley, 1998).

⁵ For a more detailed look at the underlying data and file structures of the different types of files see Mary E. S. Loomis, *Data Management and File Structures*, 2nd Ed. (Englewood Cliffs, NJ: Prentice Hall, 1989), and Michael J. Folk and Bill Zeeelick, *File Structures: A Conceptual Toolkit* (Reading, MA: Addison-Wesley, 1987).

and “after” images of data as it is altered so that an audit can be performed if the integrity of the data is questioned.

Sometimes files become so large that they are unwieldy, and much of the information in the file is no longer used. The *history file* (or archive file) stores past transactions (e.g., old customers, past orders) that are no longer needed by system users. Typically the file is stored off-line, yet it can be accessed on an as-needed basis. Other files, such as master files, can then be streamlined to include only active or very recent information.

Relational Databases

The *relational database* is the most popular kind of database for application development today. A relational database is based on collections of tables with each table having a *primary key*—a field(s) whose value is unique for every row of the table. The tables are related to each other by placing the primary key from one table into the related table as a *foreign key* (see Figure 11-3). Most *relational database management systems* (RDBMS) support *referential integrity*, or the idea of ensuring that values linking the tables together through the primary and foreign keys are valid and correctly synchronized. For example, if an order entry clerk using the tables in Figure 11-3 attempted to add order 254 for customer number 1111, he or she would have made a mistake because no customer exists in the Customer table with that number. If the RDBMS supported referential integrity, it would check the customer numbers in the Customer table; discover that the number 1111 is invalid; and return an error to the entry clerk. The clerk would then go back to the original order form and recheck the customer information. Can you imagine the problems that would occur if the RDBMS let the entry clerk add the order with the wrong information? There would be no way to track down the name of the customer for order 254.

Tables have a set number of columns and a variable number of rows that contain occurrences of data. *Structured query language* (SQL) is the standard language for accessing the data in the tables. SQL operates on complete tables, as opposed to the individual rows in the tables. Thus, a query written in SQL is applied to all of the rows in a table all at once, which is different than a lot of programming languages that manipulate data row by row. When queries must include information from more than one table, the tables first are *joined* together based on their primary key and foreign key relationships and treated as if they were one large table. Examples of RDBMS software are Microsoft Access, Oracle, DB2, and Microsoft SQL Server.

To use a relational database management system to store objects, objects must be converted so that they can be stored in a table. From a design perspective, this entails mapping a UML class diagram to a relational database schema. We describe the mapping necessary later in this chapter.

Object-Relational Databases

Object-relational database management systems (ORDBMS) are relational database management systems with extensions to handle the storage of objects in the relational table

YOUR
TURN

11-1 Student Admissions System

Pretend that you are building a Web-based system for the admissions office at your university that will be used to accept electronic applications from students. All of the data for the system will be stored in a variety of files.

Question:

- Give an example using the above system for each of the following file types: master, look-up, transaction, audit, and history. What kind of information would each file contain, and how would the file be used?

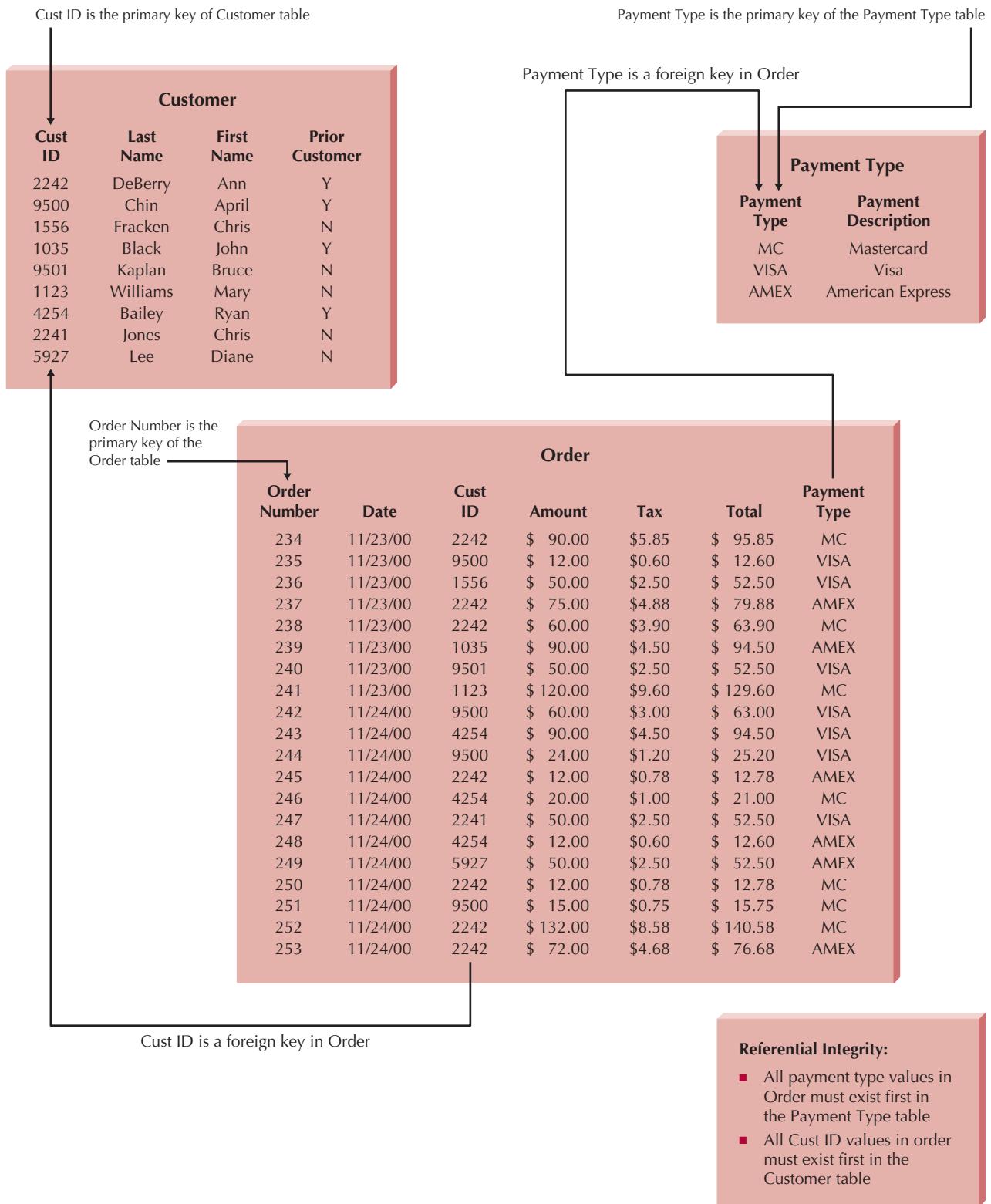


FIGURE 11-3 Relational Database

structure. This is typically done through the use of user-defined types. For example, an attribute in a table could have a data type of map, which would support storing a map. This is an example of a complex data type. In pure RDBMSs, attributes are limited to simple or atomic data types, such as integers, floats, or chars.

ORDBMSs, since they are simply extensions to their RDBMS counterpart, also have very good support for the typical data management operations that business has come to expect from RDBMSs, including an easy-to-use query language (SQL), authorization, concurrency-control, and recovery facilities. However, because SQL was designed to handle only simple data types, it too has been extended to handle complex object data. Currently, vendors deal with this issue in different manners. For example, DB2, Informix, and Oracle all have extensions that provide some level of support for objects.

Many of the ORDBMSs on the market still do not support many of the object-oriented features that can appear in an object-oriented design (e.g., inheritance). One of the problems in supporting inheritance is that inheritance support is language dependent. For example, the manner in which Smalltalk supports inheritance is different than C++'s approach, which is different than Java's approach. As such, vendors currently must support many different versions of inheritance, one for each object-oriented language, or decide on a specific version and force developers to map their object-oriented design (and implementation) to their approach. As such, like RDBMSs, a mapping from a UML class diagram to an object-relational database schema is required. We describe the mapping necessary later in this chapter.

Object-Oriented Databases

The final type of database management system that we describe is the *object-oriented database management systems (OODBMS)*. There have been two primary approaches to supporting object persistence within the OODBMS community: adding persistence extensions to an object-oriented programming language and the creation of an entirely separate database management system. In the case of OODBMS, the Object Data Management Group (ODMG) has begun the standardization process for defining (Object Definition Language, ODL), manipulating (Object Manipulating Language, OML), and querying (Object Query Language, OQL) objects in an OODBMS.⁶

With an OODBMS, collections of objects are associated with an extent. An *extent* is simply the set of instances associated with a particular class (i.e., it is the equivalent of a table in a RDBMS). Technically speaking, each instance of a class has a unique identifier assigned to it by the OODBMS: the *Object ID*. However, from a practical point of view, it is still a good idea to have a semantically meaningful primary key (even though from an OODBMS perspective this is unnecessary). Referential integrity is still very important. In an OODBMS, from the user's perspective, it looks as if the object is contained within the other object. However, the OODBMS actually keeps track of these relationships through the use of the Object ID, and as such, foreign keys are not necessary.⁷

OODBMSs provide support for some form of inheritance. However, as already discussed, inheritance tends to be language dependent. Currently, most OODBMSs are tied closely to either a particular *object-oriented programming language (OOPL)* or a set of OOPLs. Most OODBMSs originally supported either Smalltalk or C++. Today, many of the commercially available OODBMSs provide support for C++, Java, and Smalltalk.

⁶ See www.odmg.org for more information.

⁷ Depending on the storage and updating requirements, it usually is a good idea to use a foreign key in addition to the Object ID. The Object ID has no semantic meaning. As such, in the case of needing to rebuild relationships between objects, Object IDs are difficult to validate. Foreign keys, by contrast, should have some meaning outside of the DBMS.

OODBMSs also support the idea of *repeating groups* or *multivalued attributes*. These are supported through the use of *attribute sets* and *relationships sets*. RDBMSs explicitly do not allow for multivalued attributes or repeating groups. This is considered to be a violation of the first normal form (discussed later in this chapter) for relational databases. Some ORDBMSs do support repeating groups and multivalued attributes.

Up until recently, OODBMSs have mainly been used to support multimedia applications or systems that involve complex data (e.g., graphics, video, and sound). Application areas, such as computer-aided design and manufacturing (CAD/CAM), financial services, geographic information systems, health care, telecommunications, and transportation, have been the most receptive to OODBMSs. They also are becoming popular technology for supporting electronic commerce, online catalogs, and large Web multimedia applications. Examples of pure OODBMSs include Gemstone, Jasmine, O2, Objectivity, Object-Store, POET, and Versant.

Although pure OODBMS exist, most organizations currently invest in *ORDBMS* technology. The market for OODBMS is expected to grow, but its ORDBMS and RDBMS counterparts dwarf it. One reason for this situation is that there are many more experienced developers and tools in the RDBMS arena. Furthermore, relational users find that using an OODBMS comes with a fairly steep learning curve.

Selecting an Object-Persistence Format

Each of the file and database storage formats that have been presented has its strengths and weaknesses, and no one format is inherently better than the others are. In fact, sometimes a project team will choose multiple formats (e.g., a relational database for one, a file for another, and an object-oriented database for a third). Thus, it is important to understand the strengths and weaknesses of each format and when to use each one. Figure 11-4 presents a summary of the characteristics of each and the characteristics that can help identify when each type of format is more appropriate.

Major Strengths and Weaknesses The major strengths of files include that some support for sequential and random access files are normally part of an OOPL, files can be designed to be very efficient, and they are a good alternative for temporary or short-term storage. However, all file manipulation must be done through the OOPL. Files do not have any form of access control beyond that of the underlying operating system. Finally, in most cases, if files are used for permanent storage, redundant data most likely will result. This can cause many update anomalies.

RDBMS bring with them proven commercial technology. They are the leaders in the DBMS market. Furthermore, they can handle very diverse data needs. However, they typically do not handle complex data types. Therefore, all objects must be converted to a form that can be stored in tables composed of atomic or simple data. They provide no support for object-orientation. This lack of support causes an *impedance mismatch* between the objects contained in the OOPL and the data stored in the tables. An impedance mismatch refers to the amount of work, done by both the developer and DBMS, and the potential information loss that can occur when converting objects to a form that can be stored in tables.

Since ORDBMSs are typically object-oriented extensions to RDBMSs, they inherit the strengths of RDBMSs. They are based on established technologies, such as SQL, and unlike their predecessors, they can handle complex data types. However, they only provide limited support for object-orientation. The level of support varies among the vendors; therefore, ORDBMSs also suffer from the impedance mismatch problem.

OODBMSs support complex data types and have the advantage of directly supporting object-orientation. Therefore, they do not suffer from the impedance mismatch that the

	Sequential and Random Access Files	Relational DBMS	Object Relational DBMS	Object-Oriented DBMS
Major Strengths	Usually part of an object-oriented programming language Files can be designed for fast performance Good for short-term data storage	Leader in the database market Can handle diverse data needs	Based on established, proven technology, e.g., SQL Able to handle complex data	Able to handle complex data Direct support for object-orientation
Major Weaknesses	Redundant data Data must be updated using programs, i.e., no manipulation or query language No access control	Cannot handle complex data No support for object-orientation Impedance mismatch between tables and objects	Limited support for object-orientation Impedance mismatch between tables and objects	Technology is still maturing Skills are hard to find
Data Types Supported	Simple and Complex	Simple	Simple and Complex	Simple and Complex
Types of Application Systems Supported	Transaction processing	Transaction processing and decision making	Transaction processing and decision making	Transaction processing and decision making
Existing Storage Formats	Organization dependent	Organization dependent	Organization dependent	Organization dependent
Future Needs	Poor future prospects	Good future prospects	Good future prospects	Good future prospects

FIGURE 11-4 Comparison of Object Persistence Formats

previous DBMSs do. Even though the ODMG has released version 3.0 of its set of standards, the OODBMS community is still maturing. As such, this technology may still be too risky for some firms. The other major problems with OODBMS are the lack of skilled labor and the perceived steep learning curve of the RDBMS community.

Data Types Supported The first issue is the type of data that will need to be stored in the system. Most applications need to store simple data types, such as text, dates, and numbers, and all files and DBMSs are equipped to handle this kind of data. The best choice for simple data storage, however, usually is the RDBMS because the technology has matured over time and has continuously improved to handle simple data very effectively.

Increasingly, applications are incorporating complex data, such as video, images, or audio. ORDBMSs or OODBMSs are best able to handle data of this type. Complex data stored as objects can be manipulated much faster than with other storage formats.

Type of Application System There are many different kinds of application systems that can be developed. *Transaction processing systems* are designed to accept and process many simultaneous requests (e.g., order entry, distribution, and payroll). In transaction processing systems, the data continuously are updated by a large number of users, and the queries that are asked of the systems typically are predefined or targeted at a small subset of records (e.g., “List the orders that were backordered today.” Or “What products did customer #1234 order on May 12, 2001?”).

Another set of application systems are those designed to support decision making, such as *decision support systems (DSS)*, *management information systems (MIS)*, *executive information systems (EIS)*, and *expert systems (ES)*. These decision-making support systems are built to support users who need to examine large amounts of read-only historical data. The questions that they ask are often ad hoc, and they include hundreds or thousands of records at a time (e.g., “List all customers in the West region who purchased a product costing more than \$500 at least three times.” or “What products had increased sales in the summer months that have not been classified as summer merchandise?”).

Transaction processing and decision support systems thus have very different data storage needs. Transaction processing systems need data storage formats that are tuned for a lot of data updates and fast retrieval of predefined, specific questions. Files, relational databases, object-relational databases, and object-oriented databases can all support these kinds of requirements. By contrast, systems to support decision making are usually only reading data (not updating it), often in ad hoc ways. The best choices for these systems usually are RDBMSs because these formats can be configured specially for needs that may be unclear and less apt to change the data.

Existing Storage Formats The storage format should be selected primarily on the basis of the kind of data and application system being developed. However, project teams should consider the existing storage formats in the organization when making design decisions. In this way, they can better understand the technical skills that already exist and how steep the learning curve will be when the storage format is adopted. For example, a company that is familiar with RDBMS will have little problem adopting a relational database for the project, whereas an OODBMS may require substantial developer training. In the latter situation, the project team may have to plan for more time to integrate the object-oriented database with the company’s relational systems, or possibly consider moving toward an ORDBMS solution.

Future Needs Not only should a project team consider the storage technology within the company, but also it should be aware of current trends and technologies that are being used by other organizations. A large number of installations of a specific type of storage format suggest that skills and products are available to support the format. Therefore, the selection of that format is “safe.” For example, it would probably be easier and less expensive to find RDBMS expertise when implementing a system than to find help with an OODBMS.

Other Miscellaneous Criteria Other criteria that should be considered include cost, licensing issues, concurrency control, ease of use, security and access controls, version management, storage management, lock management, query management, language bindings, and APIs. We also should consider performance issues, such as cache management, insertion, deletion, retrieval, and updating of complex objects. Finally, the level of support for object-orientation

YOUR TURN

11-2 Donation Tracking System

A major public university graduates approximately 10,000 students per year, and the development office has decided to build a Web-based system that solicits and tracks donations from the university’s large alumni body. Ultimately, the development officers hope to use the information in the system to better understand the alumni giving patterns so that they can improve giving rates.

Question:

1. What kind of system is this? Does it have characteristics of more than one? What different kinds of data will this system use? On the basis of your answers, what kind of data storage format(s) do you recommend for this system?

(such as objects, single inheritance, multiple inheritance, polymorphism, encapsulation and information hiding, methods, multi-valued attributes, and repeating groups) is critical.

MAPPING PROBLEM DOMAIN OBJECTS TO OBJECT-PERSISTENCE FORMATS⁸

As described in the previous section, there are many different formats from which to choose to support object persistence. Each of the different formats can have some conversion requirements. Regardless of the object-persistence format chosen, we suggest supporting primary keys and foreign keys by adding them to the problem domain classes at this point in time. However, this does imply some additional processing required. The developer now will have to set the value for the foreign key when adding the relationship to an object. In some cases, this overhead may be too costly. In those cases, this suggestion should be ignored. In the remainder of this section, we describe how to map the problem domain classes to the different object-persistence formats. From a practical perspective, file formats are used mostly for temporary storage. As such, we do not consider them further.

We also recommend that the data management functionality specifics, such as retrieval and updating of data from the object storage, be included only in classes contained in the Data Management layer. This will ensure that the Data Management classes are dependent on the *Problem Domain classes* and not the other way around. Furthermore, this allows the design of Problem Domain classes to be independent of any specific object persistence environment, thus increasing their portability and their potential for reuse. Like our previous recommendation, this one also implies additional processing. However, the increased portability and potential for reuse realized should more than compensate for the additional processing required.

Mapping Problem Domain Objects to an OODBMS Format

If we support object persistence with an OODBMS, the mappings between the problem domain objects and the OODBMS tend to be fairly straightforward. As a starting point, we suggest that each concrete problem domain class should have a corresponding object persistence class in the OODBMS. Furthermore, there will be a data access and manipulation (DAM) class that contains the functionality required to manage the interaction between the object persistence class and the problem domain layer. For example, using the appointment system example from the previous chapters, the Patient class is associated with an OODBMS class (see Figure 11-5). The Patient class essentially will be unchanged from the analysis phase. The Patient-ODBMS class will be a new class that is dependent on the Patient class, while the Patient-DAM class will be a new class that is dependent on both the Patient class and the Patient-ODBMS class. The Patient-DAM class must be able to read from and write to the OODBMS. Otherwise, it will not be able to store and retrieve instances of the Patient class. Even though this does add overhead to the installation of the system, it allows the Problem Domain class to be independent of the OODBMS being used. As such, if at a later point in time another OODBMS or object-persistence format is adopted, only the data access and management classes will have to be modified. This approach increases both the portability and the potential for reuse of the Problem Domain classes.

Even though we are implementing the data management layer using an OODBMS, a mapping from the problem domain layer to the OODBMS classes in the data management layer may be required, depending on the level of support of inheritance in the OODBMS

⁸ The rules presented in this section are based on material in Ali Bahrami, *Object Oriented Systems Development using the Unified Modeling Language*, McGraw Hill, 1999, Michael Blaha and William Premerlani, *Object-Oriented Modeling and Design for Database Applications*, Prentice Hall, 1998, Akmal B.Chaudri and Roberto Zicari, *Succeeding with Object Databases: A Practical Look at Today's Implementations with Java and XML*, John Wiley and Sons, 2001, Peter Coad and Edward Yourdon, *Object-Oriented Design*, Yourdon Press, 1991, and Paul R. Read, Jr., *Developing Applications with Java and UML* (Boston: Addison-Wesley, 2002).

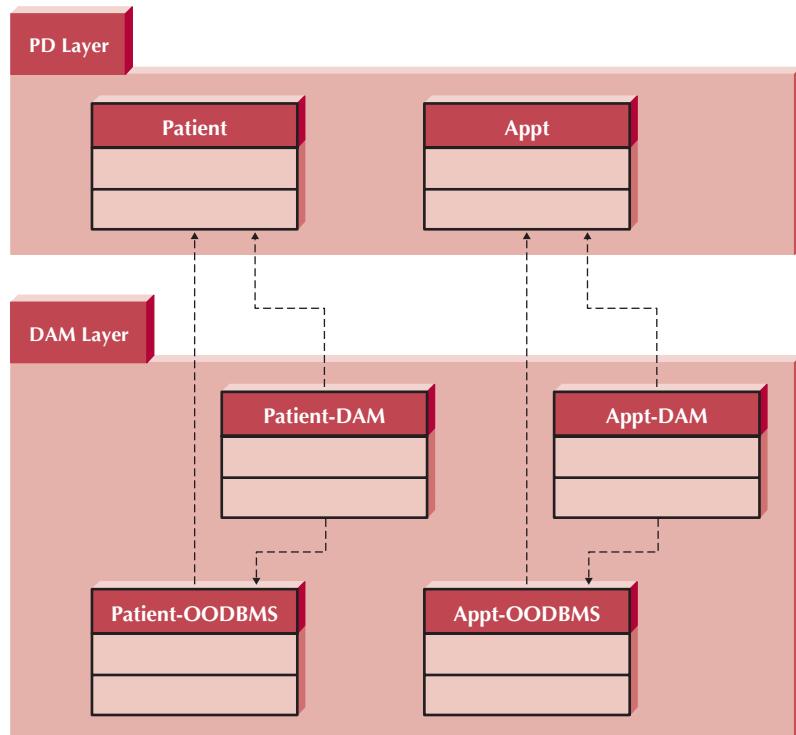


FIGURE 11-5
Appointment System
Problem Domain and
Data Access and
Management Layers

and the level of inheritance used in the problem domain classes. If multiple inheritance is used in the problem domain, but not supported by the OODBMS, then the multiple inheritance must be factored out of the OODBMS classes. For each case of multiple inheritance (i.e., more than one superclass), the following rules can be used to factor out the multiple inheritance effects in the design of the OODBMS classes.

RULE 1a: Add an attribute(s) to the OODBMS class(es) that represents the subclass(es) that will contain an Object ID of the instance stored in the OODBMS class that represents the “additional” superclass(es). This is similar in concept to a foreign key in an RDBMS. The multiplicity of this new association from the subclass to the “superclass” should be 1..1. Add an attribute(s) to the OODBMS class(es) that represents the superclass(es) that will contain an Object ID of the instance stored in the OODBMS class that represents the subclass(es). If the superclasses are concrete, that is, they can be instantiated themselves, then the multiplicity from the superclass to the subclass is 0..*, otherwise, it is 1..1. Furthermore, an exclusive-or (XOR) constraint must be added between the associations. Do this for each “additional” superclass.

OR

RULE 1b: Flatten the inheritance hierarchy of the OODBMS classes by copying the attributes and methods of the additional OODBMS superclass(es) down to all of the OODBMS subclasses and remove the additional superclass from the design.⁹

These multiple inheritance rules are very similar to those described previously in Chapter 10. Figure 11-6 demonstrates the application of the above rules. The right side of the figure portrays a set of problem domain classes that are involved in a set of multiple inheritance relationships where Class 1 inherits from both SuperClass1 and SuperClass2, and Class 2 inherits from both SuperClass2 and SuperClass3. Part A of the figure portrays the mapping

⁹ It is also a good idea to document this modification in the design so that in the future, modifications to the design can be maintained easily.

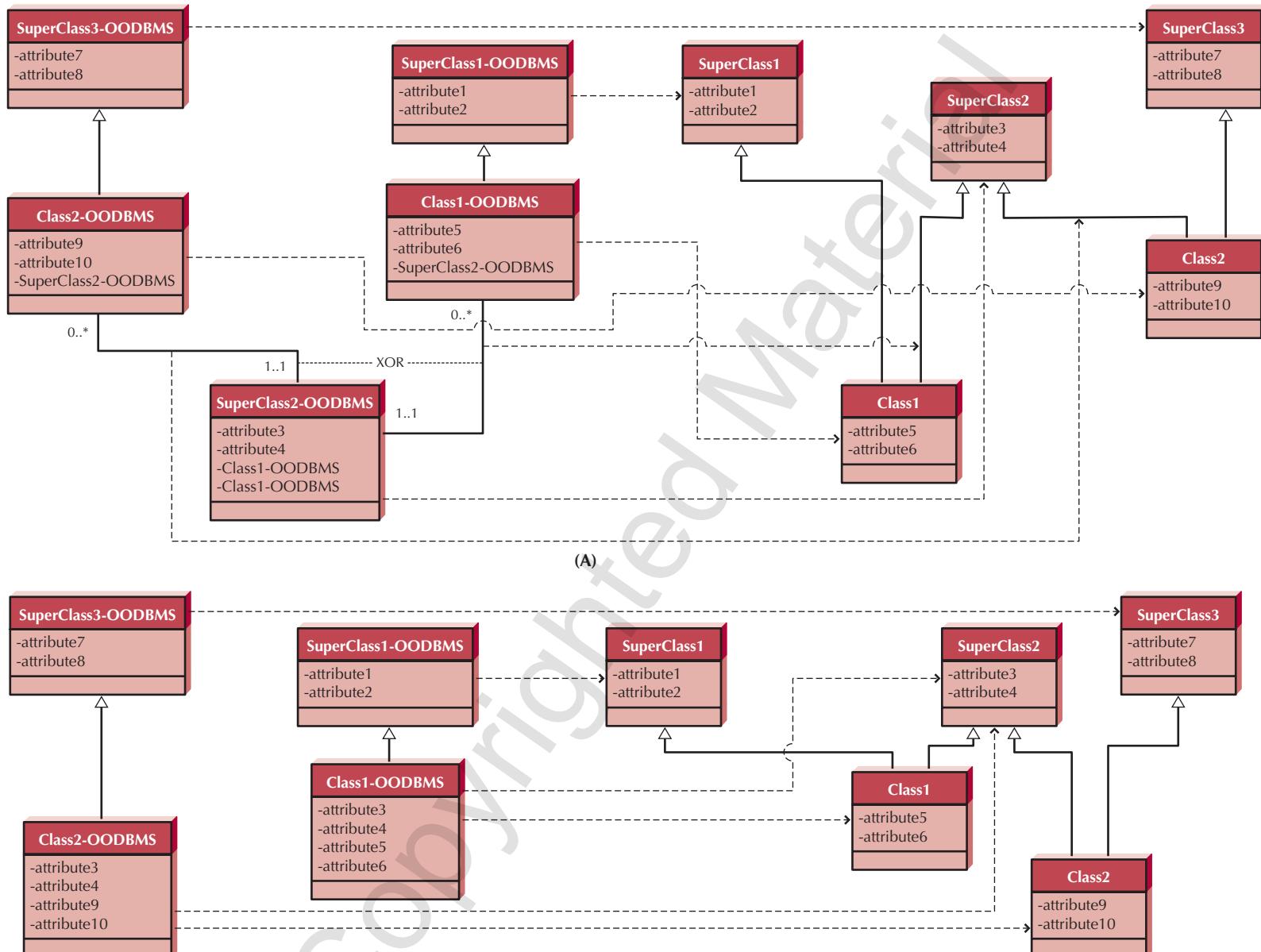


FIGURE 11-6 Mapping Problem Domain Objects to Single Inheritance-Based OODBMS

of multiple inheritance relationships into a single inheritance-based OODBMS using Rule 1a. Assuming that SuperClass2 is concrete, we apply Rule 1a to the problem domain classes, and we end up with the OODBMS classes on the left side of Part A where we have

- added an attribute to Class1-ODDBMS that represents an association with SuperClass2-ODDBMS,
- added attributes to Class2-ODDBMS that represents an association with SuperClass2-ODDBMS,
- added a pair of attributes to SuperClass2-ODDBMS that represents an association with Class1-ODDBMS and Class2-ODDBMS, for completeness sake, and
- added associations between Class2-ODDBMS and SuperClass2-ODDBMS and Class1-ODDBMS and SuperClass2-ODDBMS that have the correct multiplicities and the XOR constraint explicitly shown.

We also display the dependency relationships from the OODBMS classes to the problem domain classes. Furthermore, we illustrate the fact that the association between Class1-ODDBMS and SuperClass2-ODDBMS and the association between Class2-ODDBMS and SuperClass2-ODDBMS are based on the original factored out inheritance relationships in the problem domain classes by showing dependency relationships from the associations to the inheritance relationships.

On the other hand, if we apply Rule 1b to map the problem domain classes to a single-inheritance based OODBMS, we end up with the mapping in Part B where all of the attributes of SuperClass2 have been copied into the Class1-ODDBMS and Class2-ODDBMS classes. In this latter case, you may have to deal with the effects of inheritance conflicts (see Chapter 10).

The advantage of Rule 1a is that all problem domain classes identified during analysis are preserved in the database. This allows for maximum flexibility of maintenance of the design of the data management layer. However, Rule 1a increases the amount of message passing required in the system, and it has added processing requirements involving the XOR constraint, thus reducing the overall efficiency of the design. As such, our recommendation is to limit Rule 1a to only be applied when dealing with “extra” superclasses that are concrete because they have an independent existence in the problem domain. Use Rule 1b when they are abstract because they do not have an independent existence from the subclass.

In either case, additional processing will be required. In the first case, cascading of deletes will work, not only from the individual object to all of its elements, but also from the “superclass” instances to all of the “subclass” instances. Most OODBMSs do not support this type of deletion. However, to enforce the referential integrity of the system, it must be done. In the second case, there will be a lot of copying and pasting of the structure of the superclass to the subclasses. In the case that a modification of the structure of the superclass is required, then the modification must be cascaded to all of the subclasses. Again, most OODBMSs do not support this type of modification cascading. Therefore, the developer must address it. However, multiple inheritance is rare in most business problems. As such, in most situations the above rules will never be necessary.

When instantiating problem domain objects from OODBMS objects, additional processing also will be required. The additional processing will be in the retrieval of the OODBMS objects and taking their elements to create a problem domain object. Also, when storing the problem domain object, the conversion to a set of OODBMS objects is required. Basically speaking, anytime that an interaction takes place between the OODBMS and the system, if multiple inheritance is involved and the OODBMS only supports single inheritance, a conversion between the two formats will be required. This conversion is the purpose of the data access and manipulation classes.

**YOUR
TURN**

11-3 Dentist Office Appointment System

In the previous chapters, we have been using a dentist office appointment system as an example. Assume that you now know that the OODBMS that will be

used to support the system will only support single-inheritance. Using a class diagram, draw the design for the database.

Mapping Problem Domain Objects to an ORDBMS Format

If we support object persistence with an ORDBMS, then the mapping from the problem domain objects to the data management objects is much more involved. Depending on the level of support for object-orientation, different mapping rules are necessary. For our purposes, we assume that the ORDBMS supports Object IDs, multivalued attributes, and stored procedures. However, we assume that the ORDBMS does not provide any support for inheritance. Based on these assumptions, Figure 11-7 lists a set of rules that can be used to design the mapping from the problem domain objects to the ORDBMS-based data access and management layer tables.

Rule 1: Map all concrete problem domain classes to the ORDBMS tables. Also, if an abstract problem domain class has multiple direct subclasses, map the abstract class to an ORDBMS table.

Rule 2: Map single valued attributes to columns of the ORDBMS tables.

Rule 3: Map methods and derived attributes to stored procedures or to program modules.

Rule 4: Map single-valued aggregation and association relationships to a column that can store an Object ID. Do this for both sides of the relationship.

Rule 5: Map multi-valued attributes to a column that can contain a set of values.

Rule 6: Map repeating groups of attributes to a new table and create a one-to-many association from the original table to the new one.

Rule 7: Map multi-valued aggregation and association relationships to a column that can store a set of Object IDs. Do this for both sides of the relationship.

Rule 8: For aggregation and association relationships of mixed type (one-to-many or many-to one), on the single-valued side (1..1 or 0..1) of the relationship, add a column that can store a set of Object IDs. The values contained in this new column will be the Object IDs from the instances of the class on the multi-valued side. On the multi-valued side (1..* or 0..*), add a column that can store a single Object ID that will contain the value of the instance of the class on the single-valued side.

For generalization/inheritance relationships:

Rule 9a: Add a column(s) to the table(s) that represents the subclass(es) that will contain an Object ID of the instance stored in the table that represents the superclass. This is similar in concept to a foreign key in an RDBMS. The multiplicity of this new association from the subclass to the "superclass" should be 1..1. Add a column(s) to the table(s) that represents the superclass(es) that will contain an Object ID of the instance stored in the table that represents the subclass(es). If the superclasses are concrete, that is, they can be instantiated themselves, then the multiplicity from the superclass to the subclass is 0..*, otherwise, it is 1..1. Furthermore, an exclusive-or (XOR) constraint must be added between the associations. Do this for each superclass.

OR

Rule 9b: Flatten the inheritance hierarchy by copying the superclass attributes down to all of the subclasses and remove the superclass from the design.¹⁰

FIGURE 11-7 Mapping Problem Domain Objects to ORDBMS Schema

¹⁰ Again, it is also a good idea to document this modification in the design so that in the future, modifications to the design can be maintained easily.

First, all concrete problem domain classes must be mapped to the tables in the ORDBMS. For example in Figure 11-8, the Patient class has been mapped to Patient ORDBMS table. Notice that the Person class has been mapped also to an ORDBMS table. Even though the Person class is abstract, this mapping was done since in the complete class diagram (see Figure 7-2), the Person class had multiple direct subclasses (Employee and Patient).

Second, single-valued attributes should be mapped to columns in the ORDBMS tables. Again, referring to Figure 11-8, we see that the amount attribute of the Patient class has been included in the Patient Table class.

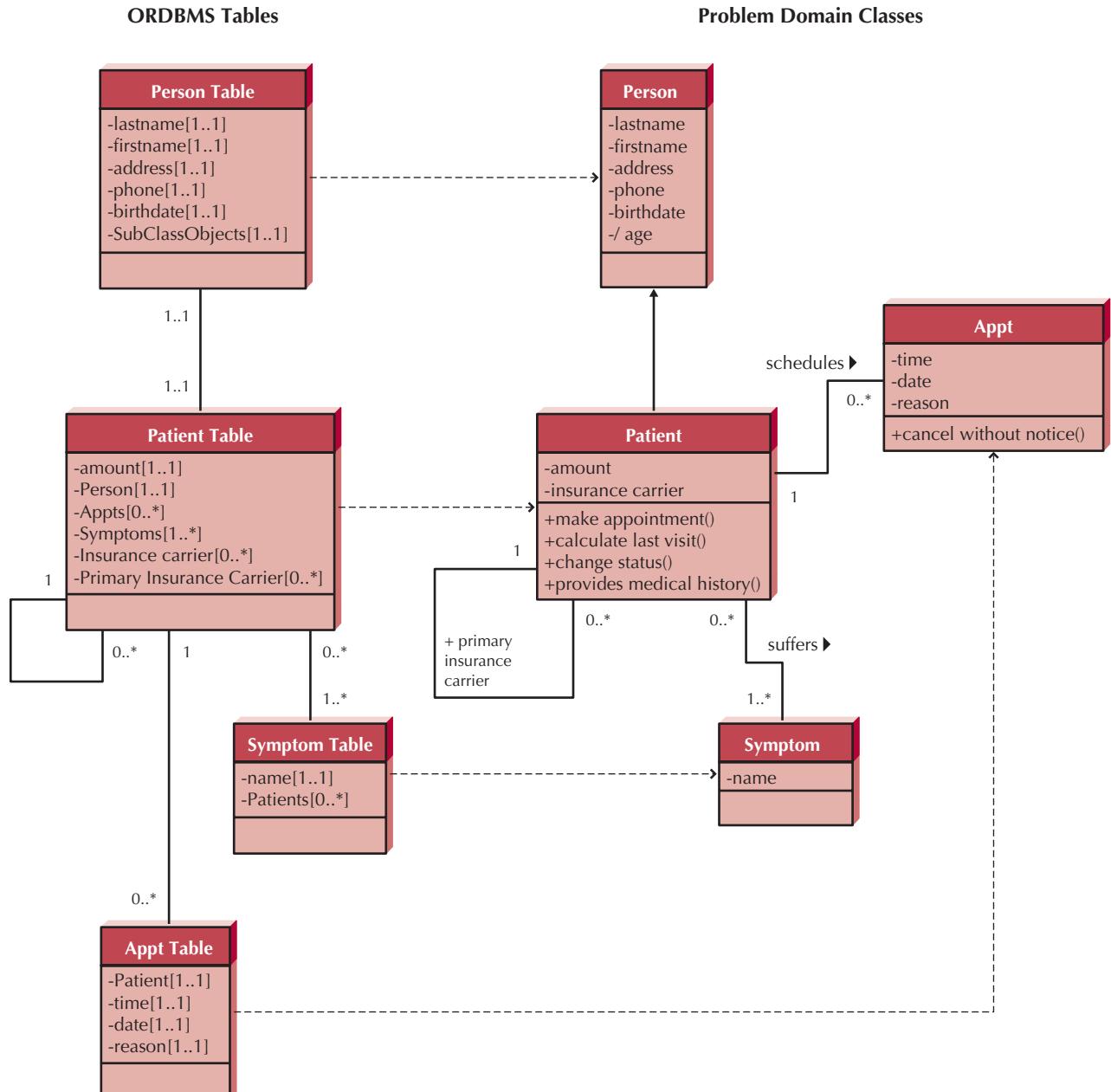


FIGURE 11-8 Mapping Problem domain Objects to ORDBMS Schema Example

Third, depending on the level of support of stored procedures, the methods and derived attributes should be mapped either to stored procedures or program modules.

Fourth, single-valued (one-to-one) aggregation and association relationships should be mapped to a column that can store an Object ID. This should be done for both sides of the relationship.

Fifth, multivalued attributes should be mapped to columns that can contain a set of values. For example in Figure 11-8, the insurance carrier attribute in the Patient class may contain multiple values because a patient may have more than one insurance carrier. As such, in the Patient table, a multiplicity has been added to the insurance carrier attribute to portray this fact.

The sixth mapping rule addresses repeating groups of attributes in a problem domain object. In this case, the repeating group of attributes should be used to create a new table in the ORDBMS. Furthermore, it may imply a missing class in the problem domain layer. Normally, when a set of attributes repeat together as a group, it implies a new class. Finally, you should create a one-to-many association from the original table to the new one.

The seventh rule supports mapping multi-valued (many-to-many) aggregation and association relationships to columns that can store a set of Object IDs. Basically, this is a combination of the fourth and fifth rules. Like the fourth rule, this should be done for both sides of the relationships. For example in Figure 11-8, the Symptom table has a multivalued attribute (Patients) that can contain multiple Object IDs to Patient Table objects, and Patient table has a multivalued attribute (Symptoms) that can contain multiple Object IDs to Symptom Table objects.

The eighth rule combines the intentions of rules 4 and 7. In this case, the rule maps one-to-many and many-to-one relationships. On the single-valued side (1..1 or 0..1) of the relationship, a column that can store a set of Object IDs from the table on the multivalued side (1..* or 0..*) of the relationship should be added. On the multi-valued side, a column should be added to the table that can store an Object ID from an instance stored in the table on the single-valued side of the relationship. For example, in Figure 11-8, the Patient table has a multivalued attribute (Appts) that can contain multiple Object IDs to Appt Table objects, while the Appt table has a single-valued attribute (Patient) that can contain an Object ID to a Patient Table object.

The ninth, and final, rule deals with the lack of support for generalization and inheritance. In this case, there are two different approaches. These approaches virtually are identical to the rules described with OODBMS object-persistence formats above. For example in Figure 11-8, the Patient table contains an attribute (Person) that can contain an Object ID for a Person Table object, and the Person table contains an attribute (SubClassObjects) that contain an Object ID for an object, in this case, stored in the Patient table. In the other case, the inheritance hierarchy is flattened.

Of course, there will be additional processing required anytime that an interaction takes place between the database and the system. Every time an object must be created, retrieved from the database, updated, or deleted, the ORDBMS object(s) must be converted to the problem domain object or vice versa. Again, this is the purpose of the data access and manipulation classes. The only other choice is to modify the problem domain objects. However, this can cause problems between the problem domain layer and the physical architecture and human computer interface layers. Generally speaking, the cost of conversion between the ORDBMS and the problem domain layer will be more than offset by the savings in development time associated with (1) the interaction between the problem domain and physical architecture and human computer interaction layers, and (2) the ease of maintenance of a semantically clean problem domain layer. In the long run, due to the conversions necessary, the development and production cost of using an OODBMS may be less than the development and production cost implementing the object persistence in an ORDBMS.

**YOUR
TURN**

11-4 Dentist Office Appointment System

In Your Turn 11-3, you created a design for the database assuming that the database would be implemented using an OODBMS that only supported single-inheritance. In this case, assume that you now know that an ORDBMS

will be used and that it does not support any inheritance. However, it does support Object IDs, multivalued attributes, and stored procedures. Using a class diagram, draw the design for the database.

Mapping Problem Domain Objects to an RDBMS Format

If we support object persistence with an RDBMS, then the mapping from the problem domain objects to the RDBMS tables is similar to the mapping to an ORDBMS. However, the assumptions made for an ORDBMS are no longer valid. Figure 11-9 lists a set of rules that can be used to design the mapping from the problem domain objects to the RDBMS-based data management layer tables.

The first four rules are basically the same set of rules that are used to map problem domain objects to ORDBMS-based data management objects. First, all concrete problem domain classes must be mapped to tables in the RDBMS. Second, single-valued attributes should be mapped to columns in the RDBMS table. Third, methods should be mapped either to stored procedures or program modules, depending on the complexity of the method. Fourth, single-valued (one-to-one) aggregation and association relationships are mapped to columns that can store the foreign keys of the related tables. This should be done for both sides of the relationship. For example in Figure 11-10, we needed to include tables in the RDBMS for the Person, Patient, Symptom, and Appt classes.

Rule 1: Map all concrete problem domain classes to the RDBMS tables. Also, if an abstract problem domain class has multiple direct subclasses, map the abstract class to a RDBMS table.

Rule 2: Map single valued attributes to columns of the tables.

Rule 3: Map methods to stored procedures or to program modules.

Rule 4: Map single-valued aggregation and association relationships to a column that can store the key of the related table, i.e., add a foreign key to the table. Do this for both sides of the relationship.

Rule 5: Map multi-valued attributes and repeating groups to new tables and create a one-to-many association from the original table to the new ones.

Rule 6: Map multi-valued aggregation and association relationships to a new associative table that relates the two original tables together. Copy the primary key from both original tables to the new associative table, i.e., add foreign keys to the table.

Rule 7: For aggregation and association relationships of mixed type, copy the primary key from the single-valued side (1..1 or 0..1) of the relationship to a new column in the table on the multi-valued side (1..* or 0..*) of the relationship that can store the key of the related table, i.e., add a foreign key to the table on the multi-valued side of the relationship.

For generalization/inheritance relationships:

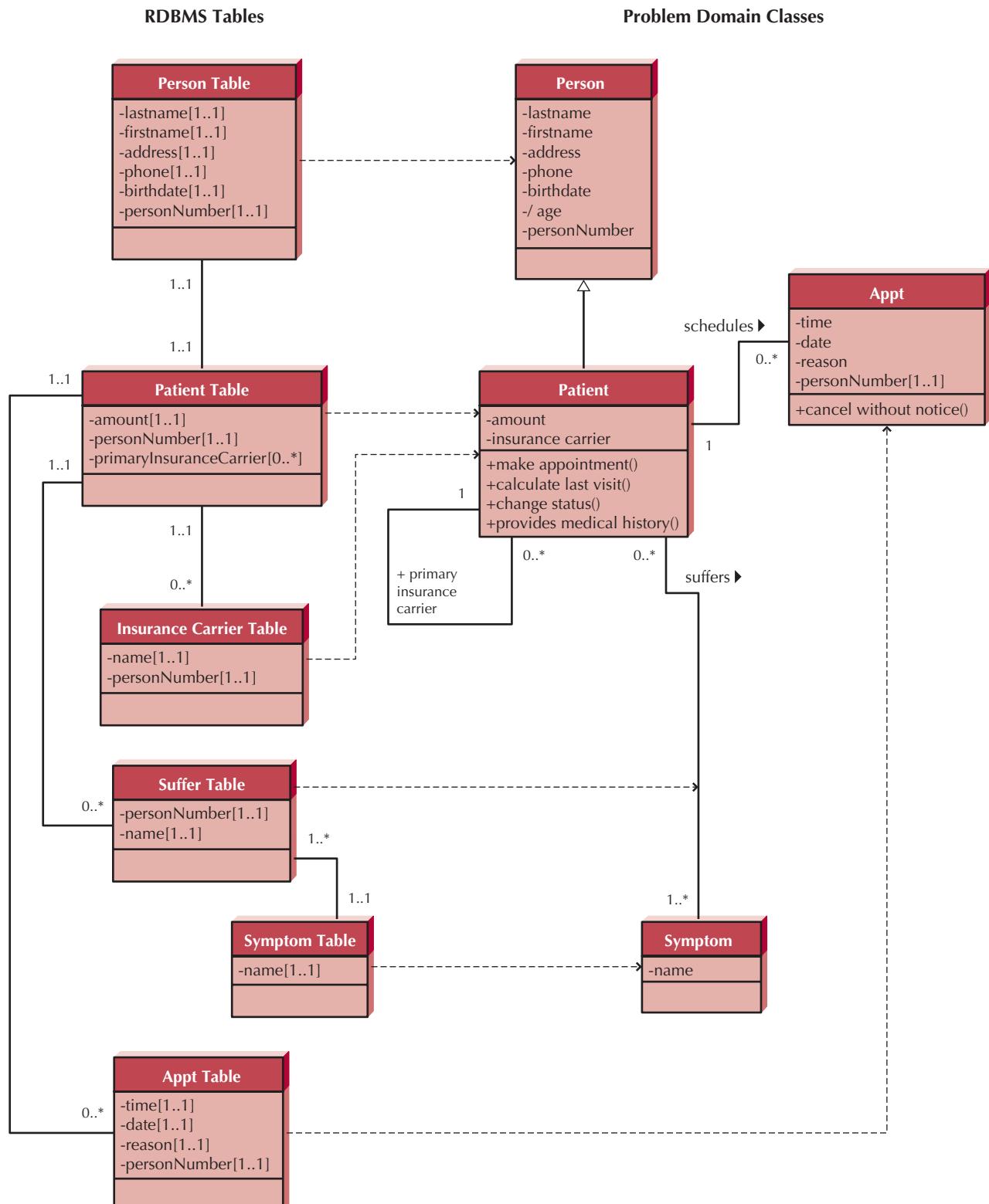
Rule 8a: Ensure that the primary key of the subclass instance is the same as the primary key of the superclass. The multiplicity of this new association from the subclass to the “superclass” should be 1..1. If the superclasses are concrete, that is, they can be instantiated themselves, then the multiplicity from the superclass to the subclass is 0..*, otherwise, it is 1..1. Furthermore, an exclusive-or (XOR) constraint must be added between the associations. Do this for each superclass.

OR

Rule 8b: Flatten the inheritance hierarchy by copying the superclass attributes down to all of the subclasses and remove the superclass from the design.¹¹

FIGURE 11-9 Mapping Problem Domain Objects to RDBMS Schema

¹¹ Again, it is also a good idea to document this modification in the design so that in the future, modifications to the design can be maintained easily.

**FIGURE 11-10** Mapping Problem Domain Objects to RDBMS Schema Example

The fifth rule addresses multi-valued attributes and repeating groups of attributes in a problem domain object. In these cases, the attributes should be used to create new tables in the RDBMS. Furthermore, like in the ORDBMS mappings, repeating groups of attributes may imply missing classes in the problem domain layer. As such, a new problem domain class may be required. Finally, you should create a one-to-many or zero-to-many association from the original table to the new one. For example, in Figure 11-10, we needed to create a new table for insurance carrier because it was possible for a patient to have more than one insurance carrier.

The sixth rule supports mapping multi-valued (many-to-many) aggregation and association relationships to a new table that relates the two original tables together. In this case, the new table should contain foreign keys back to the original tables. For example, in Figure 11-10, we needed to create a new table that represents the suffer association between the Patient and Symptom problem domain classes.

The seventh rule addresses one-to-many and many-to-one relationships. With these types of relationships, the multivalued side ($0..*$ or $1..*$) should be mapped to a column in its table that can store a foreign key back to the single-valued side ($0..1$ or $1..1$). It is possible that you have already taken care of this situation since we recommended earlier that you include both primary and foreign key attributes in the problem domain classes. In the case of Figure 11-10, we had already added the primary key from the Patient class to the Appt class as a foreign key (see personNumber). However, in the case of the reflexive relationship, primary insurance carrier, associated with the Patient class, we need to add a new attribute (primaryInsuranceCarrier) to be able to store the relationship.

The eighth, and final, rule deals with the lack of support for generalization and inheritance. Like in the case of an ORDBMS, there are two different approaches. These approaches virtually are identical to the rules described with OODBMS and ORDBMS object-persistence formats above. The first approach is to add a column to each table that represents a subclass for each of the concrete superclasses of the subclass. Essentially, this is ensuring that the primary key of the subclass is the same as the primary key for the superclass. If you had previously added the primary and foreign keys to the problem domain objects, as we recommended above, then you do not have to do anything else. The primary keys of the tables will be used to join back together the instances stored in the tables that represent each of the pieces of the problem domain object. Conversely, the inheritance hierarchy can be flattened and the rules (Rules 1 through 7) reapplied.

As in the case of the ORDBMS approach, additional processing will be required anytime that an interaction takes place between the database and the system. Every time an object must be created, retrieved from the database, updated, or deleted, the mapping between the problem domain and the RDBMS must be used to convert between the two different formats. In this case, a great deal of additional processing will be required. However, from a practical point of view, it is more likely that you will use a RDBMS for storage of objects than the other approaches because RDBMSs are by far the most popular format in the marketplace. As such, we will focus on how to optimize the RDBMS format for object persistence in the next section of this chapter.

OPTIMIZING RDBMS-BASED OBJECT STORAGE

Once the object-persistence format is selected, the second step is to optimize the object persistence for processing efficiency. The methods of optimization will vary based on the format that you select; however, the basic concepts will remain the same. Once you understand how to optimize a particular type of object persistence, you will have some idea as to how to approach the optimization of other formats. This section focuses on the optimization of the most popular storage format: relational databases.

**YOUR
TURN**

11-5 Dentist Office Appointment System

In Your Turn 11-3, you created a design for the database assuming that the database would be implemented using an OODBMS that only supported single-inheritance. And in Your Turn 11-4, you created a design for the database assuming that the database would be implemented using

an ORDBMS that did not support any inheritance, but did support Object IDs, multi-valued attributes, and stored procedures. In this case, you should assume that the system will be supported by a RDBMS. Using a class diagram, draw the design for the database.

There are two primary dimensions in which to optimize a relational database: for storage efficiency and for speed of access. Unfortunately, these two goals often conflict because the best design for access speed may take up a great deal of storage space as compared to other less speedy designs. This section describes how to optimize the object persistence for storage efficiency using a process called normalization. The next section presents design techniques, such as denormalization and indexing, which can speed up the performance of the system. Ultimately, the project team will go through a series of trade-offs until the ideal balance of the two optimization dimensions is reached. Finally, the project team must estimate the size of the data storage needed to ensure there is enough capacity on the server(s).

Optimizing Storage Efficiency

The most efficient tables in a relational database in terms of storage space have no redundant data and very few null values because the presence of these suggest that space is being wasted (and more data to store means higher data storage hardware costs). For example, notice that the table in Figure 11-11 repeats customer information, such as name and state, each time a customer places an order, and it contains many null values in the product related columns. These nulls occur whenever a customer places an order for less than three items (the maximum number on an order).

In addition to wasting space, redundancy and null values also allow more room for error and increase the likelihood that problems will arise with the integrity of the data. What if customer 1035 moved from Maryland to Georgia? In the case of Figure 11-11, a program must be written to ensure that all instances of that customer are updated to show “GA” as the new state of residence. If some of the instances are overlooked, then the table will contain an *update anomaly* whereby some of the records contain the correctly updated value for state and other records contain the old information.

Nulls threaten data integrity because they are difficult to interpret. A blank value in the Order table’s product fields could mean (1) the customer did not want more than one or two products on his or her order, (2) the operator forgot to enter in all three products on the order, or (3) the customer cancelled part of the order and the products were deleted by the operator. It is impossible to be sure of the actual meaning of the nulls.

For both of these reasons—wasted storage space and data integrity threats—project teams should remove redundancy and nulls from the table. During the design phase, the class diagram is used to examine the design of the RDBMS tables (e.g., see Figure 11-10) and optimize it for storage efficiency. If you follow the modeling instructions and guidelines that were presented in Chapter 7, you will have little trouble creating a design that is highly optimized in this way because a well-formed logical object model does not contain redundancy or many null values.

Sometimes, however, a project team needs to start with a model that was poorly constructed or with one that was created for files or a non-relational type of format. In these

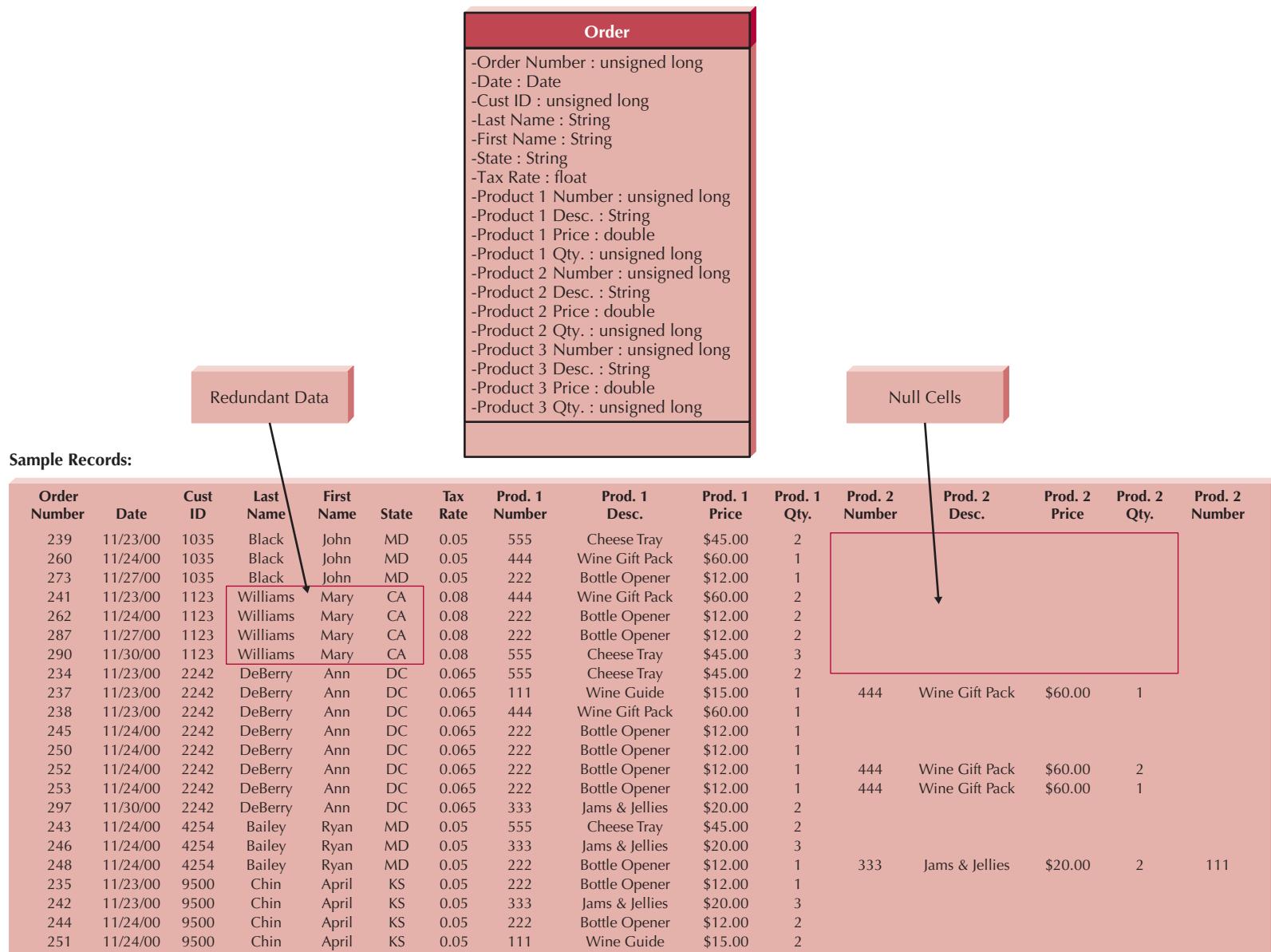


FIGURE 11-11 Optimizing Storage

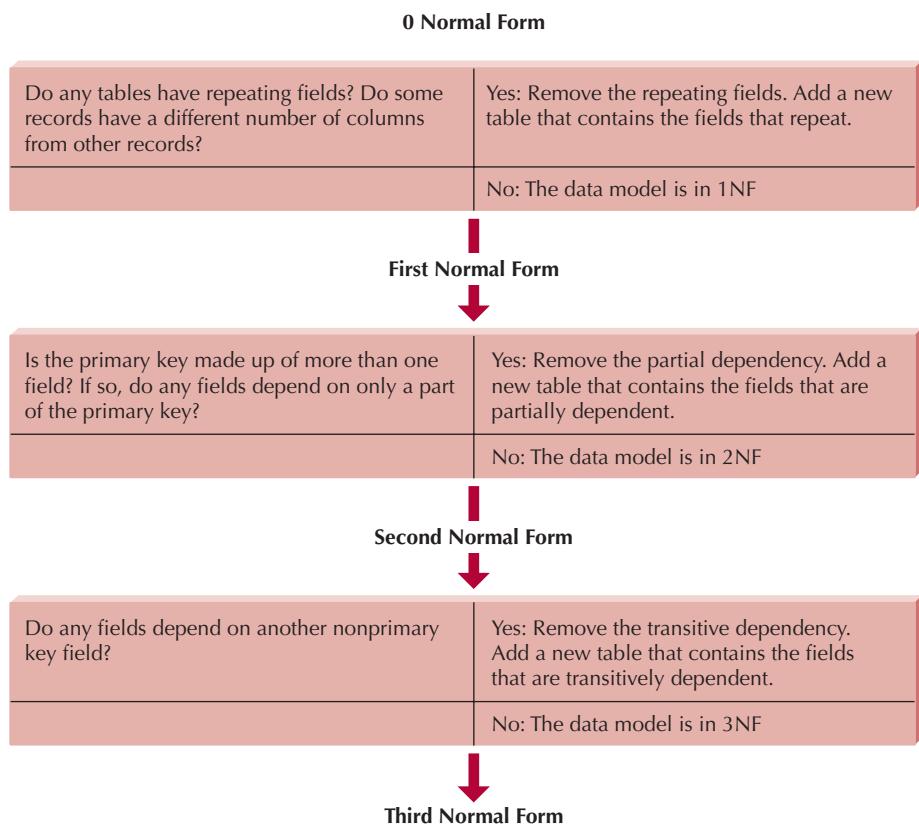


FIGURE 11-12
The Steps of
Normalization

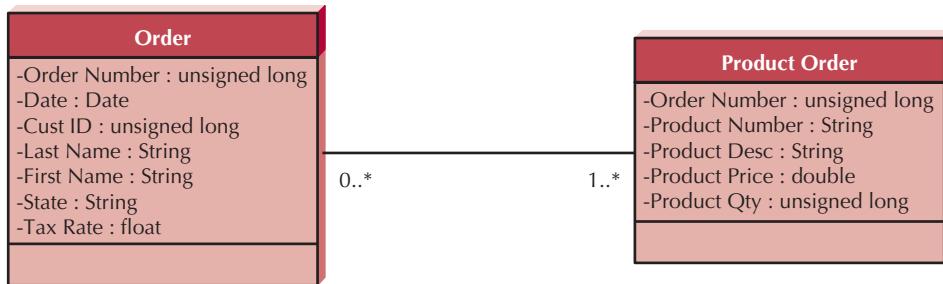
cases, the project team should follow a series of steps that serve to check the model for storage efficiency. These steps make up a process called normalization.¹²

Normalization is a process whereby a series of rules are applied to the RDBMS tables to determine how well formed they are (see Figure 11-12). These rules help analysts identify tables that are not represented correctly. Here, we describe three normalization rules that are applied regularly in practice. Figure 11-11 shows a model in 0 Normal Form, which is an unnormalized model before the normalization rules have been applied.

A model is in *first normal form (1NF)* if it does not lead to *multi-valued fields*, fields that allow a set of values to be stored, or *repeating fields*, which are fields that repeat within a table to capture multiple values. The rule for 1NF says that a table must contain the same number of columns (i.e., fields) and that all of the columns must contain a single value. Notice that the model in Figure 11-11 violates 1NF because it causes product number, description, price, and quantity to repeat three times for each order in the table. The resulting table has many records that contain nulls in the product-related columns, and orders are limited to three products because there is no room to store information for more.

A much more efficient design (and one that conforms to 1NF) leads to a separate table to hold the repeating information; to do this, we create a separate table on the model to capture product order information. A zero-to-many relationship then would exist between the two tables. As shown in Figure 11-13, the new design eliminates nulls from the Order table and supports an unlimited number of products that can be associated with an order.

¹² Normalization also can be performed on the Problem Domain layer. However, the normalization process only should be used on the Problem Domain layer to uncover missing classes. Otherwise, optimizations that have nothing to do with the semantics of the problem domain can creep into the Problem Domain layer.

Revised Model:

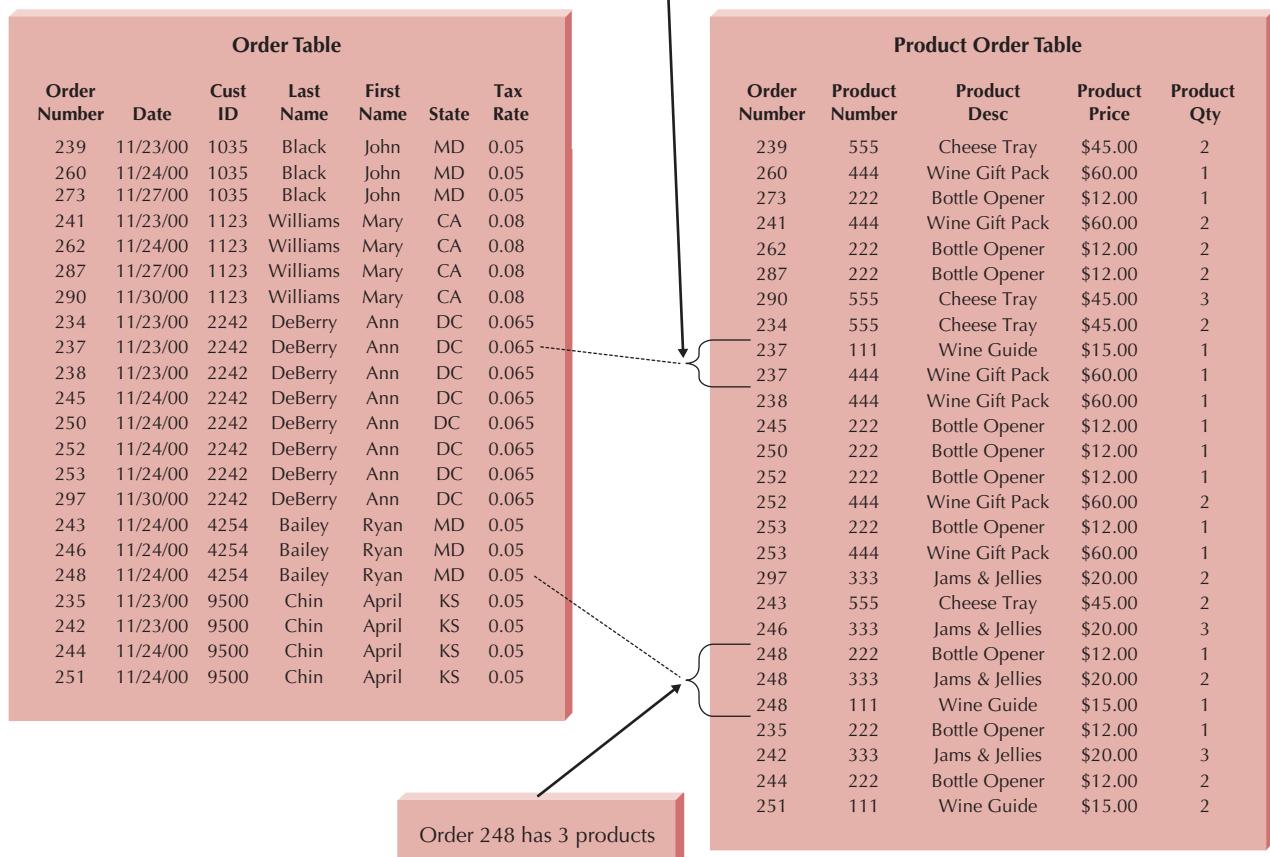
Note: Order Number will serve as part of the primary key of Order

Note: Cust ID also will serve as part of the primary key of Order

Note: Order Number will serve as part of the primary key of Product Order

Note: Product Number will serve as part of the primary key of Product Order

Note: Order Number also will serve as a foreign key in Product Order

Sample Records:**FIGURE 11-13 1NF: Remove Repeating Fields**

Second normal form (2NF) requires first that the data model is in 1NF and second that the data model leads to tables containing fields that are dependent on a *whole* primary key. This means that the primary key value for each record can determine the value for all of the other fields in the record. Sometimes fields depend only on part of the primary key (i.e., *partial dependency*), and these fields belong in another table.

For example, in the new Product Order table that was created in Figure 11-13, the primary key is a combination of the order number and product number, but the product description and price attributes are dependent only upon product number. In other words, by knowing product number, you can identify the product description and price. However, knowledge of the order number and product number is required to identify the quantity. To rectify this violation of 2NF, a table is created to store product information, and the description and price attributes are moved into the new table. Now, product description is stored only once for each instance of a product number as opposed to many times (every time a product is placed on an order).

A second violation of 2NF occurs in the Order table: customer first name and last name are dependent only upon the customer ID, not the whole key (Cust ID and Order number). As a result, every time the customer ID appears in the Order table, the names also appear. A much more economical way of storing the data is to create a Customer table with the Customer ID as the primary key and the other customer-related fields (i.e., last name and first name) listed only once within the appropriate record. Figure 11-14 illustrates how the model would look when placed in 2NF.

Third normal form (3NF) occurs when a model is in both 1NF and 2NF and in the resulting tables none of the fields are dependent on non-primary key fields (i.e., *transitive dependency*). Figure 11-14 contains a violation of 3NF: the tax rate on the order depends upon the state to which the order is being sent. The solution involves creating another table that contains state abbreviations serving as the primary key and the tax rate as a regular field. Figure 11-15 presents the end results of applying the steps of normalization to the original model from Figure 11-11.

Optimizing Data Access Speed

After you have optimized the design of the object storage for efficiency, the end result is that data is spread out across a number of tables. When data from multiple tables need to be accessed or queried, the tables must be first joined together. For example, before a user can print out a list of the customer names associated with orders, first the Customer and Order tables need to be joined together based on the customer number field (see Figure 11-15).

YOUR TURN

11-6 Normalizing a Student Activity File

Pretend that you have been asked to build a system that tracks student involvement in activities around campus. You have been given a file with information that needs to be imported into the system, and the file contains the following fields:

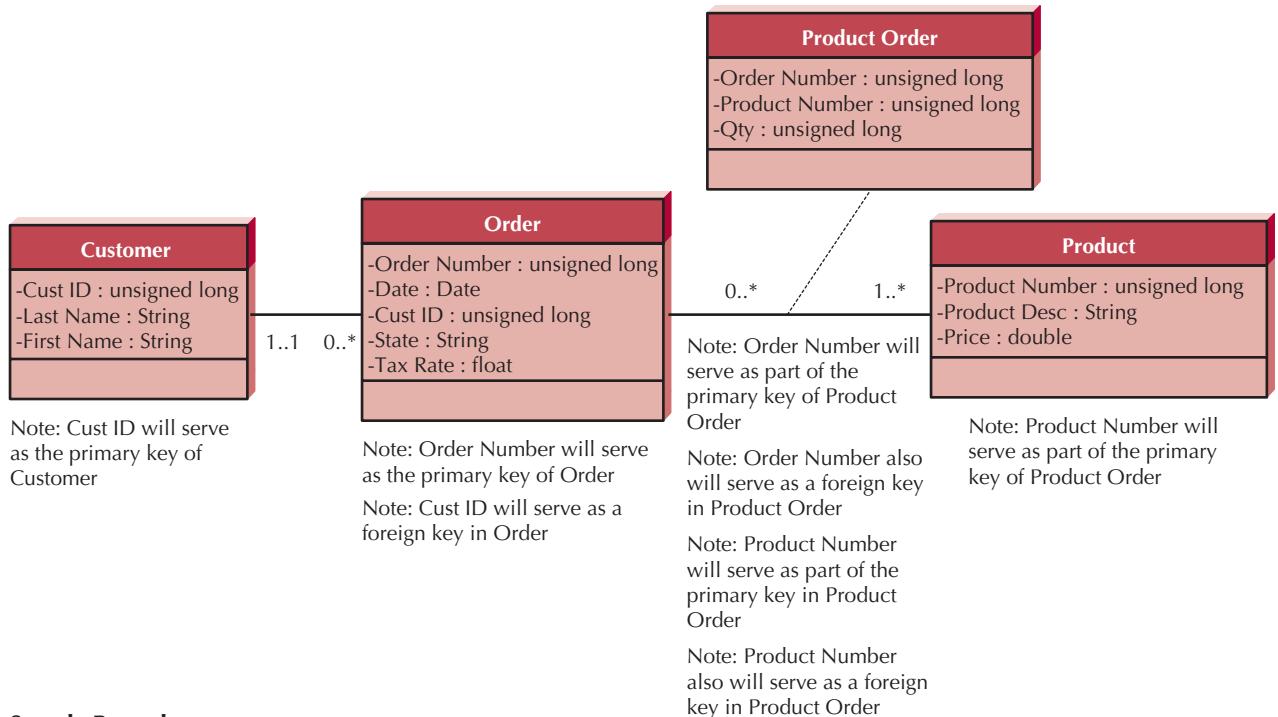
student social security number
student last name

activity 1 start date
activity 2 code

student first name
student advisor name
student advisor phone
activity 1 code
activity 1 description

activity 2 description
activity 2 start date
activity 3 code
activity 3 description
activity 3 start date

Normalize the file. Show how the logical data model would change at each step.

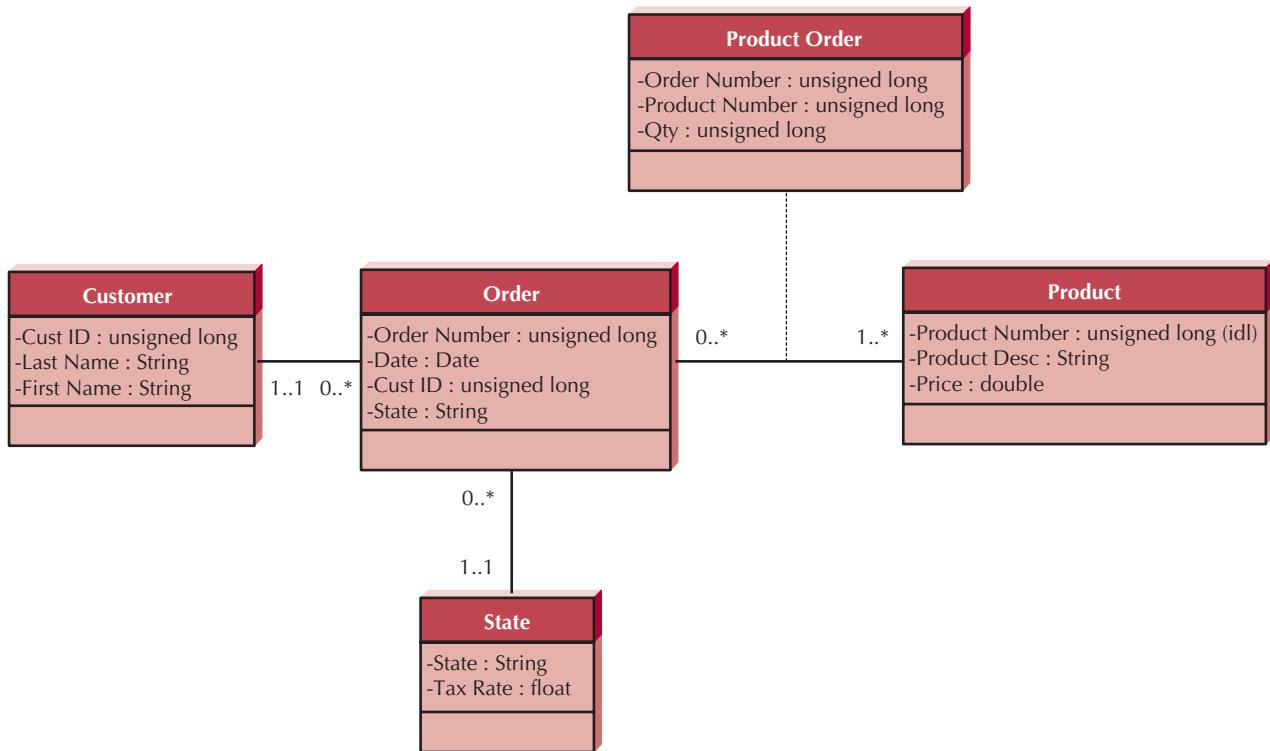
**Sample Records:**

Customer Table			Order Table				Product Order Table			Product Table		
Cust ID	Last Name	First Name	Order Number	Date	Cust ID	State	Order Number	Product Number	Product Qty	Product Number	Product Desc	Product Price
1035	Black	John	239	11/23/00	1035	MD	239	555	2	111	Wine Guide	\$15.00
1123	Williams	Mary	260	11/24/00	1035	MD	260	444	1	222	Bottle Opener	\$12.00
2242	DeBerry	Ann	273	11/27/00	1035	MD	273	222	1	333	Jams & Jellies	\$20.00
4254	Bailey	Ryan	241	11/23/00	1123	CA	241	444	2	444	Wine Gift Pack	\$60.00
9500	Chin	April	262	11/24/00	1123	CA	262	222	2	555	Cheese Tray	\$45.00
			287	11/27/00	1123	CA	287	222	2			
			290	11/30/00	1123	CA	290	555	3			
			234	11/23/00	2242	DC	234	555	2			
			237	11/23/00	2242	DC	237	111	1			
			238	11/23/00	2242	DC	238	444	1			
			245	11/24/00	2242	DC	245	222	1			
			250	11/24/00	2242	DC	250	222	1			
			252	11/24/00	2242	DC	252	222	1			
			253	11/24/00	2242	DC	253	444	2			
			297	11/30/00	2242	DC	252	222	1			
			243	11/24/00	4254	MD	253	444	1			
			246	11/24/00	4254	MD	253	222	1			
			248	11/24/00	4254	MD	243	555	2			
			235	11/23/00	9500	KS	246	333	3			
			242	11/23/00	9500	KS	248	222	1			
			244	11/24/00	9500	KS	248	333	2			
			251	11/24/00	9500	KS	248	111	1			
							235	222	1			
							242	333	3			
							244	222	2			
							251	111	2			

Notes:

- Last Name and First Name was moved to the Customer table to eliminate redundancy
- Product Desc and Price was moved to the Product table to eliminate redundancy

FIGURE 11-14 2NF Partial Dependencies Removed

**FIGURE 11-15** 3NF Normalized Model

Only then can both the order and customer information be included in the query's output. Joins can take a lot of time, especially if the tables are large or if many tables are involved.

Consider a system that stores information about 10,000 different products, 25,000 customers, and 100,000 orders, each averaging three products per order. If an analyst wanted to investigate whether there were regional differences in music preferences, he or she would need to combine all of the tables to be able to look at products that have been ordered while knowing the state of the customers placing the orders. A query of this information would result in a huge table with 300,000 rows (i.e., the number of products that have been ordered) and 11 columns (the total number of columns from all of the tables combined).

The project team can use several techniques that to try to speed up access to the data including denormalization, clustering, and indexing.

Denormalization After the object storage is optimized, the project team may decide to denormalize, or add redundancy back into the design. *Denormalization* reduces the number of joins that need to be performed in a query, thus speeding up access. Figure 11-16 shows a denormalized model for customer orders. The customer last name was added back into the Order table since the project team learned during the analysis phase that queries about orders usually require the customer last name field. Instead of joining the Order table repeatedly to the Customer table, the system now needs to access only the Order table because it contains all of the relevant information.

Denormalization should be applied sparingly for the reasons described in the previous section, but it is ideal in situations in which information is queried frequently yet updated rarely. There are three cases in which you may rely upon denormalization to reduce joins and

improve performance. First, denormalization can be applied in the case of lookup tables, which are tables that contain descriptions of values (e.g., a table of product descriptions or a table of payment types). Because descriptions of codes rarely change, it may be more efficient to include the description along with its respective code in the main table to eliminate the need to join the lookup table each time a query is performed (see Figure 11-17a).

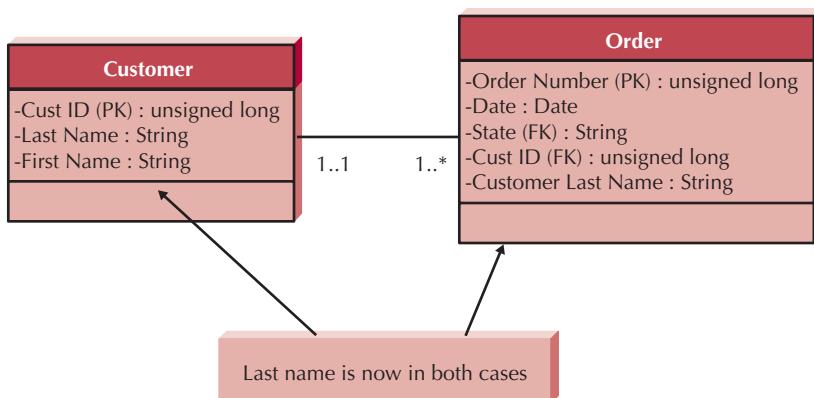
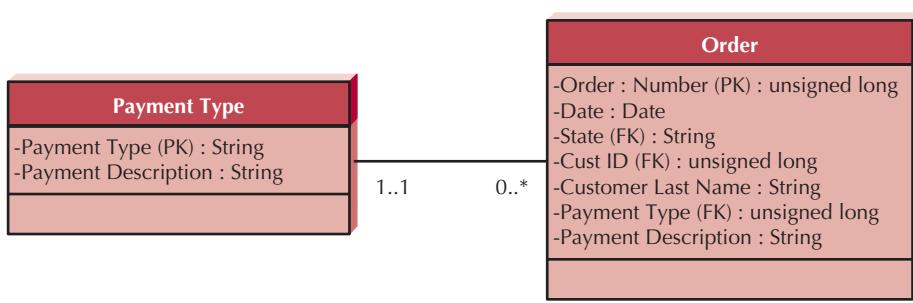
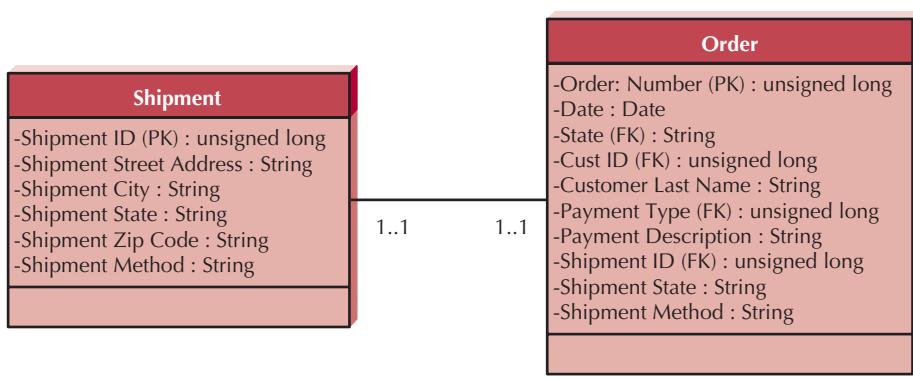


FIGURE 11-16
Denormalized Physical Data Model
(FK = foreign key;
PK = primary key)



(A)
Notice that the payment description field appears in both Payment Type and Order.



(B)
Notice that the shipment state and shipment method are included in both Shipment and Order.

FIGURE 11-17
Denormalization Situations (FK = foreign key;
PK = primary key); (a) Lookup table; (b) One-to One Relationship

Second, one-to-one relationships are good candidates for denormalization. Although logically two tables should be separated, from a practical standpoint the information from both tables may regularly be accessed together. Think about an order and its shipping information. Logically, it may make sense to separate the attributes related to shipping into a separate table, but as a result the queries regarding shipping likely will always need a join to the Order table. If the project team finds that certain shipping information like state and shipping method are needed when orders are accessed, they may decide to combine the tables or include some shipping attributes in the Order table (see Figure 11-17b).

Third, at times it will be more efficient to include a parent entity's attributes in its child entity on the physical data model. For example, consider the Customer and Order tables in Figure 11-16, which share a one-to-many relationship, with Customer as the parent and Order as the child. If queries regarding orders continuously require customer information, the most popular customer fields can be placed in Order to reduce the required joins to the Customer table as was done with Customer Last Name.

Clustering Speed of access also is influenced by the way that the data is retrieved. Think about going shopping in a grocery store. If you have a list of items to buy, but you are unfamiliar with the store's layout, then you need to walk down every aisle to make sure that you don't miss anything from your list. Likewise, if records are arranged on a hard disk in no particular order (or in an order that is irrelevant to your data needs), then any query of the records results in a *table scan* in which the DBMS has to access every row in the table before retrieving the result set. Table scans are the most inefficient of data retrieval methods.

One way to improve access speed is to reduce the number of times that the storage medium needs to be accessed during a transaction. One method is to *cluster* records together physically so that similar records are stored close together. With *intra-file clustering*, like records in the table are stored together in some way, such as in order by primary key or, in the case of a grocery store, by item type. Thus, whenever a query looks for records, it can go directly to the right spot on the hard disk (or other storage medium) because it knows in what order the records are stored, just as we can walk directly to the bread aisle to pick up a loaf of bread. *Inter-file clustering* combines records from more than one table that typically are retrieved together. For example, if customer information is usually accessed with the related order information, then the records from the two tables may be physically stored in a way that preserves the customer-order relationship. Returning to the grocery store scenario, an inter-file cluster would be similar to storing peanut butter, jelly, and bread next to each other in the same aisle because they are usually purchased together, not because they are similar types of items. Of course, each table can have only one clustering strategy because the records can be arranged physically in only one way.

Indexing A time saver that you are familiar with is an index located in the back of a textbook, which points you directly to the page or pages that contain your topic of interest. Think of how long it would take you to find all of the times that "relational database"

YOUR TURN

11-7 Denormalizing a Student Activity File

Consider the logical data model that you created for Your Turn 11-6. Examine the model and describe possible opportunities for denormalization. How would you

change the physical data model for this file, and what are the benefits of your changes?

appears in this textbook if you didn't have the index to rely on! An *index* in data storage is like an index in the back of a textbook; it is a mini table that contains values from one or more columns in a table and the location of the values within the table. Instead of paging through the entire textbook, you can move directly to the right pages and get the information you need. Indexes are one of the most important ways to improve database performance. Whenever you have performance problems, the first place to look is an index.

A query can use an index to find the locations of only those records that are included in the query answer, and a table can have an unlimited number of indexes. Figure 11-18 shows an index that orders records by payment type. A query that searches for all of the customers who used American Express can use this index to find the locations of the records that contain American Express as the payment type without having to scan the entire Order table.

Project teams can make indexes perform even faster by placing them into the main memory of the data storage hardware. Retrieving information from memory is much faster than from another storage medium like a hard disk—think about how much faster it is to retrieve a phone number that you have memorized versus one that you need to look up in a phone book. Similarly, when a database has an index in memory, it can locate records very, very quickly.

Of course, indexes require overhead in that they take up space on the storage medium. Also, they need to be updated as records in tables are inserted, deleted, or changed. Thus, although indexes lead to faster access to the data, they slow down the update process. In general, you should create indexes sparingly for transaction systems or systems that require a lot of updates, but should apply indexes generously when designing systems for decision support (see Figure 11-19).

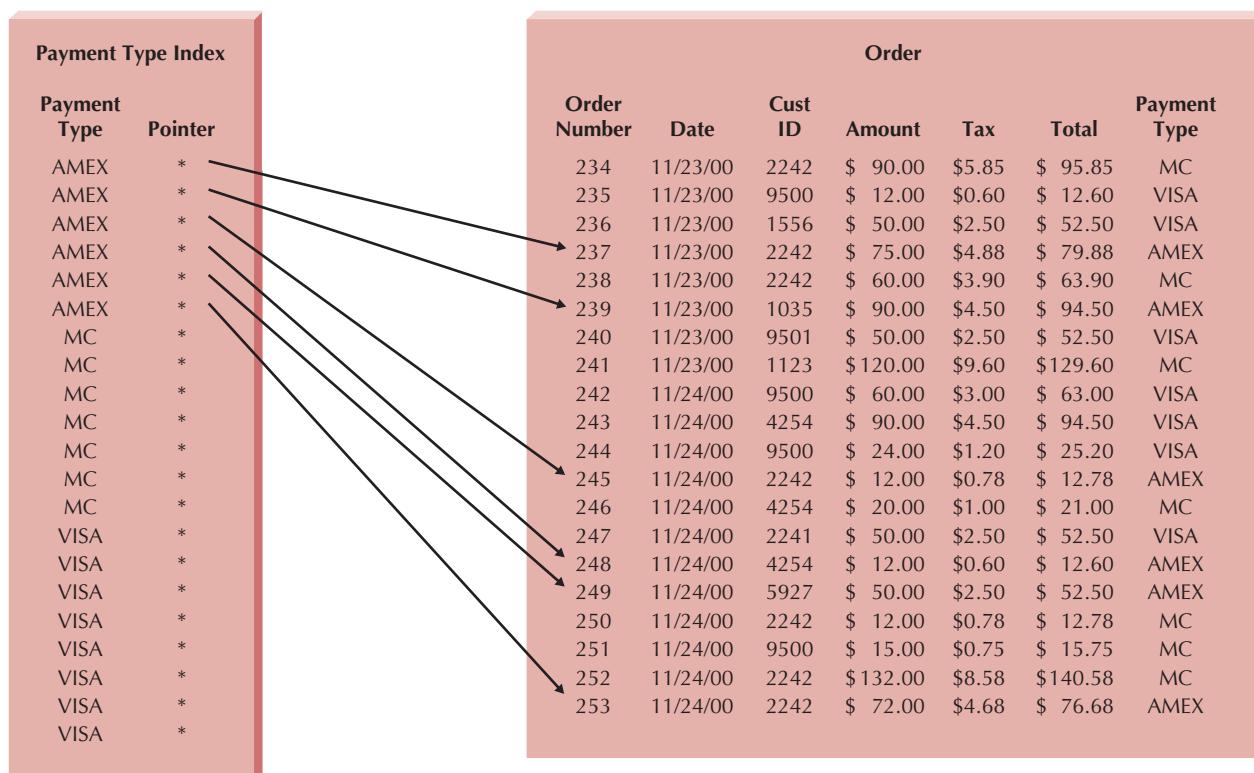


FIGURE 11-18 Payment Type Index

CONCEPTS

11-A Mail Order Index

IN ACTION

A Virginia-based mail-order company sends out approximately 25 million catalogs each year using a Customer table with 10 million names. Although the primary key of the Customer table is customer identification number, the table also contains an index of Customer Last Name. Most people who call to place orders remember their last name, not their customer identification number, so this index is used frequently.

An employee of the company explained that indexes are critical to reasonable response times. A fairly complicated query was written to locate customers by the state

in which they lived, and it took more than three weeks to return an answer. A customer state index was created, and that same query provided a response in 20 minutes: that's 1,512 times faster!

Question:

1. As an analyst, how can you make sure that the proper indexes have been put in place so that users are not waiting for weeks to receive the answers to their questions?

FIGURE 11-19
Guidelines for Creating Indexes

- Use indexes sparingly for transaction systems.
- Use many indexes to increase response times in decision support systems.
- For each table, create a unique index that is based on the primary key.
- For each table, create an index that is based on the foreign key to improve the performance of joins.
- Create an index for fields that are used frequently for grouping, sorting, or criteria.

Estimating Data Storage Size

Even if you have denormalized your physical data model, clustered records, and created indexes appropriately, the system will perform poorly if the database server cannot handle its volume of data. Therefore, one last way to plan for good performance is to apply *volumetrics*, which means estimating the amount of data that the hardware will need to support. You can incorporate your estimates into the database server hardware specification to make sure that the database hardware is sufficient for the project's needs. The size of the database is based on the amount of raw data in the tables and the overhead requirements of the DBMS. To estimate size you will need to have a good understanding of the initial size of your data as well as its expected growth rate over time.

Raw data refers to all of the data that are stored within the tables of the database, and it is calculated based on a bottom-up approach. First, write down the estimated average width for each column (field) in the table and sum the values for a total record size (see Figure 11-20). For example, if a variable width LAST NAME column is assigned a width of 20 characters, you can enter 13 as the average character width of the column. In Figure 11-20, the estimated record size is 49.

Next, calculate the overhead for the table as a percentage of each record. *Overhead* includes the room needed by the DBMS to support such functions as administrative actions and indexes, and it should be assigned based on past experience, recommendations from technology vendors, or parameters that are built into software that was written to calculate volumetrics. For example, your DBMS vendor may recommend that you allocate 30 percent of the records' raw data size for overhead storage space, creating a total record size of 63.7 in the Figure 11-20 example.

Field	Average Size
Order Number	8
Date	7
Cust ID	4
Last Name	13
First Name	9
State	2
Amount	4
Tax Rate	2
Record Size	49
Overhead	30%
Total Record Size	63.7
Initial Table Size	50,000
Initial Table Volume	3,185,000
Growth Rate/Month	1,000
Table Volume @ 3 years	5,478,200

FIGURE 11-20
Calculating Volumetrics

Finally, record the number of initial records that will be loaded into the table, as well as the expected growth per month. This information should have been collected during the analysis phase. According to Figure 11-20, the initial space required by the first table is 3,185,000, and future sizes can be projected based on the growth figure. These steps are repeated for each table to get a total size for the entire database.

Many CASE tools will provide you with database size information based on how you set up the object persistence and they will calculate volumetrics estimates automatically. Ultimately, the size of the database needs to be shared with the design team so that the proper technology can be put in place to support the system's data and potential performance problems can be addressed long before they affect the success of the system.

DESIGNING DATA ACCESS AND MANIPULATION CLASSES

The final step in developing the data management layer is to design the *data access and manipulation classes* that act as a translator between the object persistence and the problem domain objects. As such, they should always be capable of at least reading and writing both the object persistence and problem domain objects. As described earlier, and in Chapter 9, the object persistence classes are derived from the concrete problem domain classes while the data access and manipulation classes are dependent on both the object persistence and problem domain classes.

Depending on the application, a simple rule to follow is that there should be one data access and manipulation class for each concrete problem domain class. In some cases, it might make sense to create data access and manipulation classes associated with the human computer interaction classes (see Chapter 12). However, this creates a dependency from the data management layer to the human computer interaction layer. Adding this additional complexity to the design of the system normally is not recommended.

Returning to the ORDBMS solution for the Appointment system example (see Figure 11-8), we see that we have four problem domain classes and four ORDBMS tables. Following the previous rule, the data access and management classes are rather simple. They only have to support a one-to-one translation between the concrete problem domain classes and the ORDBMS tables (see Figure 11-21). Since the Person problem domain class is an abstract class, only three data access and manipulation classes are required: Patient-DAM, Symptom-DAM, and Appt-DAM. However, the process to create an instance of the Patient problem domain class can be fairly complicated. The Patient-DAM class may have to be able to retrieve information from all four ORDBMS tables. To accomplish this, the Patient-DAM class retrieves the information from the Patient table. Using the Object-IDs stored in the attribute values associated with the Person, Appts, and Symptoms attributes, the remaining information required to create an instance of Patient is easily retrieved by the Patient-DAM class.

In the case of using an RDBMS to provide persistence, the data access and manipulation classes tend to become more complex. For example, in the Appointment system, there are still

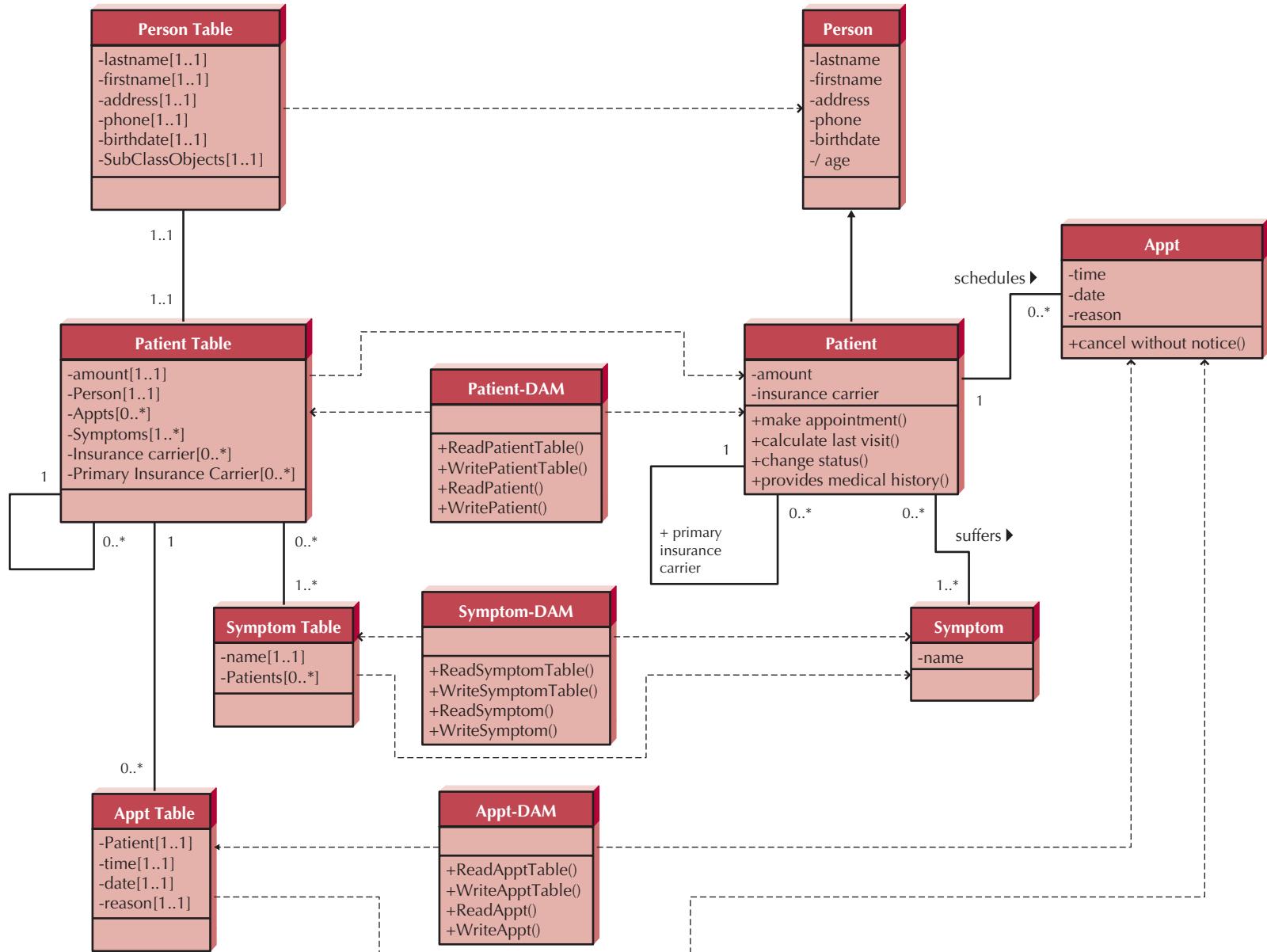


FIGURE 11-21 Mapping Problem Domain Objects to ORDBMS Using Data Access and Management Classes

four problem domain classes but, due to the limitations of RDBMSs, we have to support six RDBMS tables (see Figure 11-10). The data access and manipulation class for the Appt problem domain class and the Appt RDBMS table is no different than those supported for the ORDBMS solution (see Figures 11-21 and 11-22). However, due to the multivalued attributes and relationships associated with the Patient and Symptom problem domain classes, the mappings to the RDBMS tables were more complicated. As such, the number of dependencies from the data access and manipulation classes (Patient-DAM and Symptom-DAM) to the RDBMS tables (Patient table, Insurance Carrier table, Suffer table, and the Symptom table) has increased. Furthermore, since the Patient problem domain class is associated with the other three problem domain classes, the actual retrieval of all information necessary to create an instance of the Patient class could involve joining together information from all six RDBMS tables. To accomplish this, the Patient-DAM class must first retrieve information from the Patient table, Insurance Carrier table, Suffer table, and the Appt table. Since the primary key of the Patient table and the Person table are identical, the Patient-DAM class can either directly retrieve the information from the Person table, or they can join the information together using the personNumber attributes of the two tables, which act as both primary and foreign keys. Finally, using the information contained in the Suffer table, the information in the Symptom table can be retrieved also. Obviously, the further we get from the object-oriented problem domain class representation, the more work must be performed. However, as in the case of the ORDBMS example, notice that absolutely no modifications were made to the problem domain classes. Therefore, the data access and manipulation classes again have prevented data management functionality from creeping into the problem domain classes.

APPLYING THE CONCEPTS AT CD SELECTIONS

The CD Selections Internet sales system needs to present CD information effectively to users and to capture order data. Alec Adams, senior systems analyst and project manager for the Internet sales system, recognized that these goals were dependent upon a good design of the data management layer for the new application. He approached the design of the data management layer in four steps: by selecting the object-persistence format, mapping the problem domain classes to the selected format, optimizing the selected format for processing efficiency, and designing the data access and manipulation classes.

Select Object-Persistence Format

The project team met to discuss two issues that would drive the object-persistence format selection: what kind of objects would be in the system and how they would be used. Using a whiteboard, they listed the ideas that are presented in Figure 11-23. The project team agreed that the bulk of the data in the system would be the text and numbers that are exchanged with Web users regarding customers and orders. A relational database would be able to handle the data effectively, and the technology would be well received at CD Selections because relational technology is already in place throughout the organization.

However, they realized that relational technology was not optimized to handle complex data, such as the images, sound, and video that the marketing facet of the system ultimately will require. Alec asked Brian, one of the staff members on the Internet sales system project, to look into relational databases that offered object add-on products (i.e., an RDBMS that could become an ORDBMS). It might be possible for the team to invest in a RDBMS foundation and then upgrade to an ORDBMS version of the same product. However, in the meantime, Alec decided to store sample clips using a random file. This way, they could still deliver the system as envisioned while keeping the technology requirements reasonable.

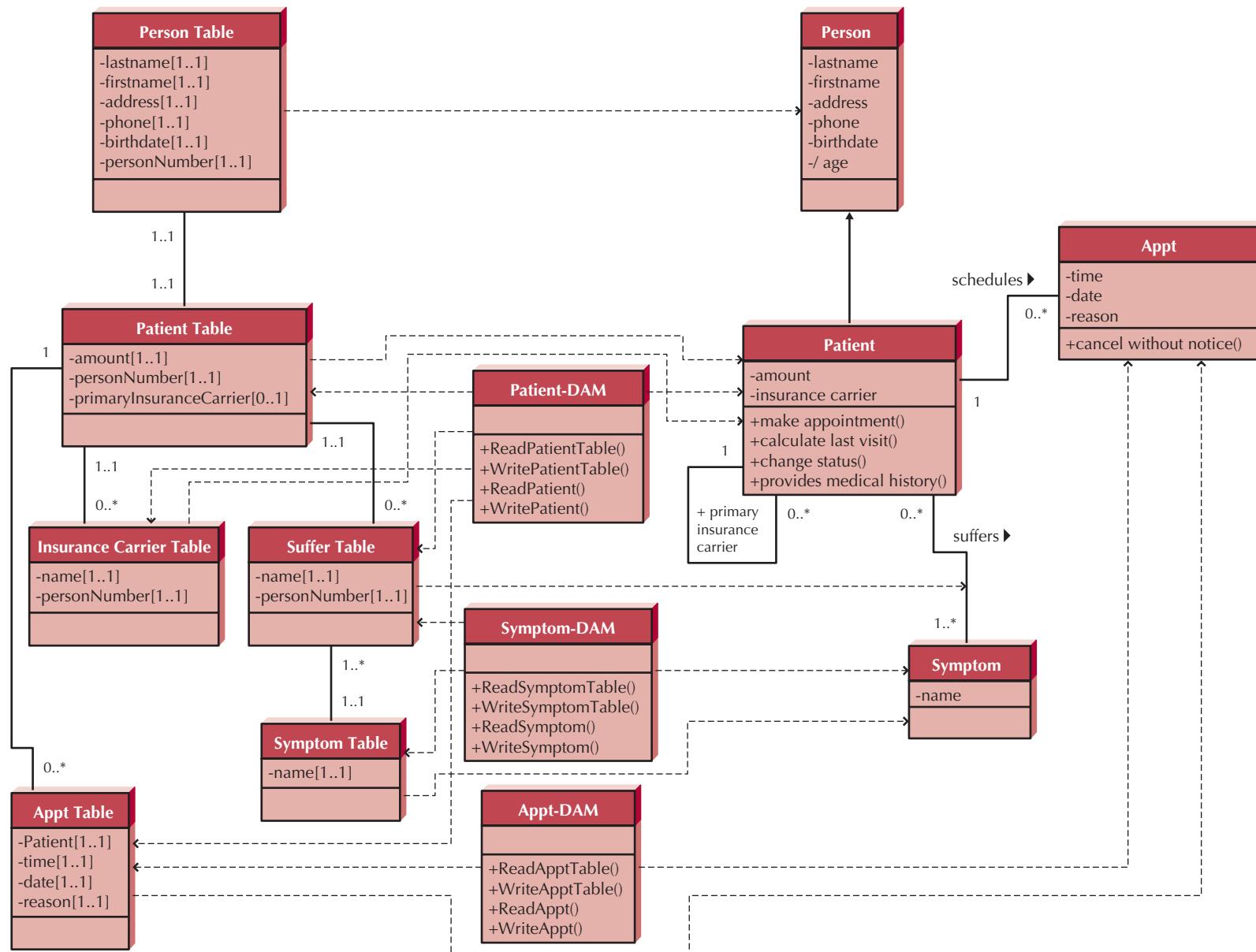


FIGURE 11-22 Mapping Problem Domain Objects to RDBMS Using Data Access and Management Classes

Data	Type	Use	Suggested Format
Customer information	Simple (mostly text)	Transactions	Relational
Order Information	Simple (text and numbers)	Transactions	Relational
Marketing Information	Both simple and complex (eventually the system will contain audio clips, video, etc.)	Transactions	Object add-on?
Information that will be exchanged with the Distribution System	Simple text, formatted specifically for importing into the Distribution System	Transactions	Transaction file
Temporary Information	The Web component will likely need to hold information for temporary periods of time. (e.g., the shopping card will store order information before the order is actually placed)	Transactions	Transaction file

FIGURE 11-23
Types of Data in Internet Sales System

The team noted that it also must design two transaction files to handle the interface with the distribution system and the Web shopping cart program. The Internet sales system will regularly download order information to the distribution system using a transaction file containing all the required information for that system. Also, the team must design the file that stores temporary order information on the Web server as customers shop through the Web site. The file would contain the fields that ultimately would be transferred to an order object.

Of course, Alec realized that other data needs might arise over time, but he felt confident that the major data issues were identified (e.g., like the capability to handle complex data) and that the design of the data management layer would be based on the proper storage technologies.

Map Problem Domain Objects to Object-Persistence Format

Based on the decision to use an RDBMS and a random file to store the problem domain objects, Alec asked Brian to create an object-persistence design. To begin, Brian first reviewed the current class and package diagrams for the evolving Internet sales system (see Figures 9-8, 10-17, and 10-21). Focusing on Figures 10-17 and 10-21, Brian began applying the appropriate mapping rules (see Figure 11-9). Based on Rule 1, Brian identified 12 problem domain classes that needed to have their objects stored; as such, Brian created 11 RDBMS tables and 1 file to represent these objects. These included Credit Card Center table, Customer table, Individual table, Organizational table, Order table, Order Item table, CD table, Vendor table, Mkt Info table, Review table, Artist Info table, and Sample Clip File. He also created a set of tentative primary keys for each of the tables and the file. Based on the fact that the objects in the Search Package (see Figures 9-8 and 10-17), Search Req, and CD List, are only temporary, Brian decided that there was no real need to address them at this point in the design.

Using Rule 4, Brian identified the need for the CD table and the Mkt Info table to both have each other's primary key stored as a foreign key to the other table. Upon further reflection, Brian reasoned that because an actual instance of marketing information was only going to be associated with a single instance of CD and vice versa, he could have merged the two tables together. However, the team had earlier decided to

keep them separate, so he decided to simply use the primary key of the CD table as the primary key of the Mkt Info table.

While reviewing the current set of attributes for each of the tables, John (another member of the data management layer design team) suggested that the team had left out the idea of a CD containing a set of tracks. As such, they added a multivalued attribute, tracks, to the CD problem domain class. However, Brian then pointed out that when they apply Rule 5 to the CD class, they really needed to factor the tracks attribute out as a separate table. Furthermore, as the team discussed the track attribute further, it was decided to include it as a problem domain class also.

Next, the data management layer team applied Rule 6 to the evolving object-persistence design. In doing this, Susan (another member of the data management layer design team) pointed out that the checks relationship between the Customer and Credit Card Center problem domain classes was a multivalued association. Therefore, it needed its own table in the relational database. Not to be upstaged by Susan, John immediately pointed out that Rule 7 was applicable to 8 associations: Customer places Order, Order includes Order Item, Order Item contains CD, Vendor distributes CD, Mkt Info promotes CD, CD contains Tracks, and the three aggregation associations with the Mkt Info class (Review, Artist Info, and Sample Clip). As such, quite a few primary keys had to be copied into the relevant tables as foreign keys (e.g., the primary key of the Customer table had to be copied to the Order table). Can you identify the others?

Finally, Susan suggested the solution to the inheritance problem because RDBMSs do not support inheritance. She pointed out that when applying Rule 8a to the Customer superclass and the Individual and Organizational subclasses, the primary key of the Customer table also had to be copied to the tables representing the subclasses. Furthermore, she pointed out that an exclusive-or (XOR) condition existed between the two subclasses.

Based on all of the suggestions and hard work accomplished by the data management layer team, Brian was able to create a design of the object persistence for the Internet sales system (see Figure 11-24).

Optimize Object Persistence and Estimate its Size

After the meeting, Alec asked Brian to stay behind to discuss the data management layer model. Now that the team had a good idea of the type of object-persistence formats that would be used, they were ready for the third step: optimizing the design for performance efficiency. Since Brian was the analyst in charge of the data management layer, Alec wanted to discuss with him whether the model was optimized for storage efficiency. He also needed this done before the team discussed access speed issues.

Brian assured Alec that the current object persistence model was in third normal form. He was confident of this because the project team followed the modeling guidelines that lead to a well-formed model.

Brian then asked about the file formats for the two transaction files identified in the earlier meeting. Alec suggested that he normalize the files to better understand the various tables that would be involved in the import procedure. Figure 11-25 shows the initial file layout for the Distribution System import file as well as the steps that were taken as Brian applied each normalization rule.

The next step for the design of the data management layer was to optimize the design for access speed. Alec met with the analysts on the data management layer design team and talked about the techniques that were available to speed up access. Together the team listed all of the data that will be supported by the Internet Sales System and discussed how all of the data would be used. They developed the strategy that is laid out in Figure 11-26 to identify the specific techniques to put in place.

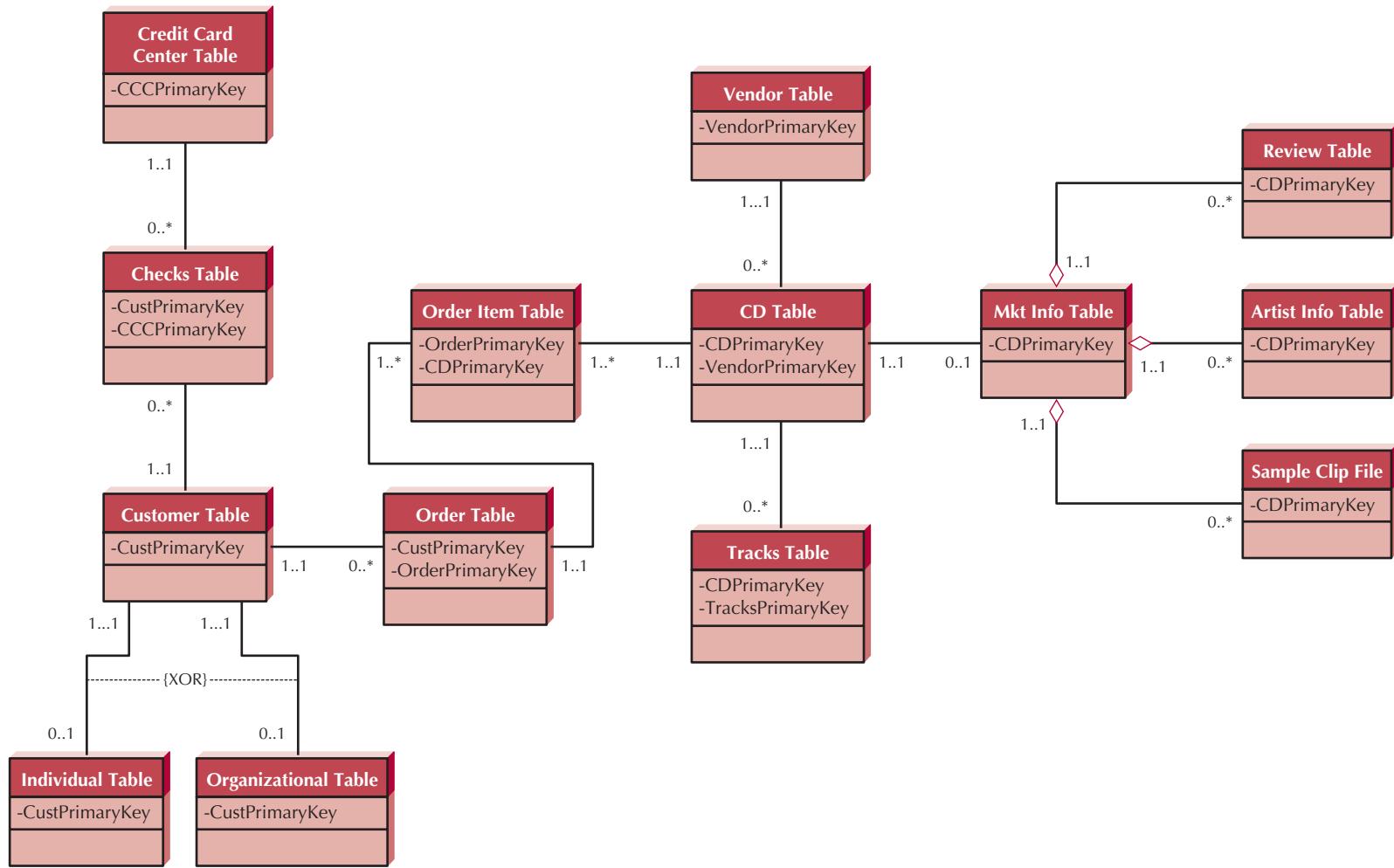


FIGURE 11-24 Internet Sales System Object Persistence Design

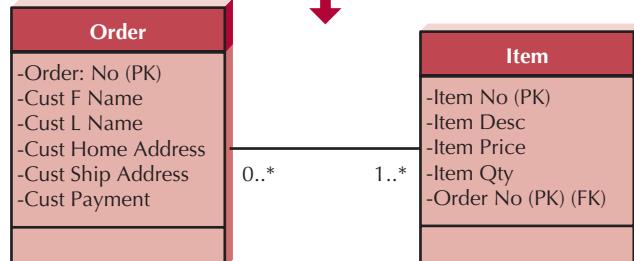
0NF:

Order Number	Cust Fname	Cust Lname	Cust Home Address	Cust Ship Address	Cust Pay	Item No*	Item Desc*	Item Price*	Item Qty*
A (7)	A (20)	A (20)	A (150)	A (150)	9999.99	A (7)	A (20)	9999.99	9999

*Item No, Item Desc, Item Price, and Item Qty repeats four (4) times.

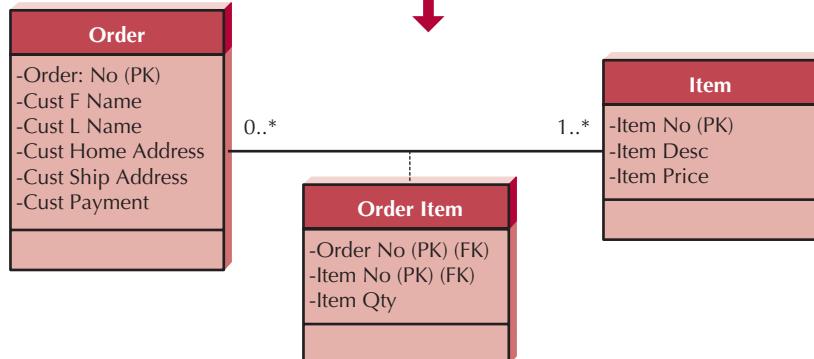
Remove repeating groups of items and place them in a separate Item entity.

1NF:



There is one partial dependency in the above data model since Item Qty is dependent on the whole primary key while Item Desc and Item Price are dependent only on Item No.

2NF:



Remove transitive dependencies. Customer Home Address is dependent on Cust F Name and Cust L Name (and these do not serve as the identifier for the Order entity); therefore, a Customer entity is added to contain customer information. Order then contains only order information and necessary foreign keys.

3NF:

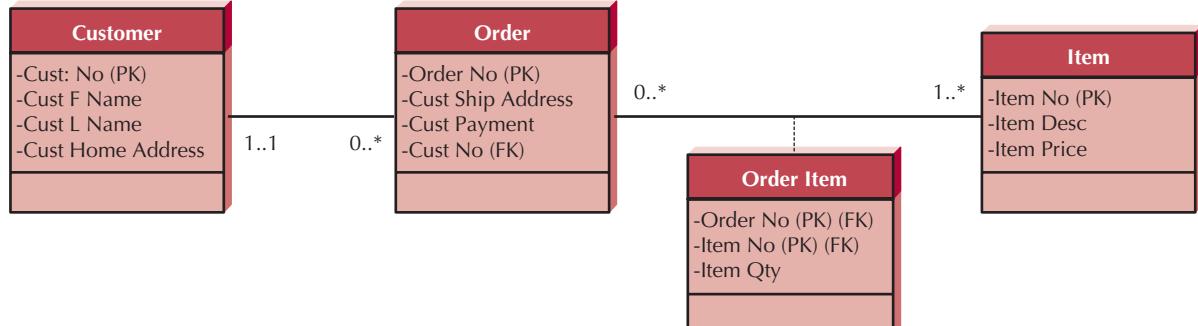


FIGURE 11-25 Distribution System Import File Normalization Process

Target	Comments	Suggestions to Improve Data Access Speed
All tables	Basic table manipulation	<ul style="list-style-type: none"> Investigate if records should be clustered physically by primary key Create indexes for primary keys Create indexes for foreign key fields
All tables	Sorts and Grouping	<ul style="list-style-type: none"> Create indexes for fields that are frequently sorted or grouped
CD information	Users will need to search CD information by title, artist, and category	<ul style="list-style-type: none"> Create indexes for CD title, artist, and category
Order Information	Operators should be able to locate information about a particular customer's order	<ul style="list-style-type: none"> Create an index in the Order table for orders by customer name
Entire Physical Model	Investigate denormalization opportunities for all fields that are not updated very often	<ul style="list-style-type: none"> Investigate one-to-one relationships Investigate lookup tables Investigate one-to-many relationships

FIGURE 11-26
Internet Sales System Performance

Ultimately, clustering strategies, indexes, and denormalization decisions were applied to the physical data model, and a volumetrics report was run from the CASE tool to estimate the initial and projected size of the database. The report suggested that an initial storage capacity of about 450 megabytes would be needed for the expected one-year life of the first version of the system. Additional storage capacity would be needed for the second version that would include sound files for samples of the songs, but for the moment not much storage would be needed.

Alec gave the estimates to the analyst in charge of managing the server hardware acquisition so that the person could make sure that the technology could handle the expected volume of data for the Internet sales system. The estimates would also go to the DBMS vendor during the implementation of the software so that the DBMS could be configured properly.

Data Access and Manipulation Class Design

The final step in designing the data management layer was to develop the design of the data access and manipulation classes that would act as translators between the object persistence and the problem domain classes. Since the CD package (see Figures 9-8 and 10-21) was the most important package, Alec asked Brian to complete the design of the data management layer for the CD package and report back to Alec when he was finished. Upon reviewing the concrete problem domain classes in the CD package, Brian realized that he needed to have seven data access and manipulation classes; one for each concrete problem domain class. These classes would be fully dependent on their related problem domain classes. Next, Brian mapped the data access and manipulation classes down to the RDBMS tables and the random file associated with storing the objects. In this case, there were six RDBMS tables and one random file. Again, the data access and manipulation classes are dependent on the object-persistence format. Figure 11-27 depicts the data management layer and problem domain layer for the CD package of the Internet sales system.

Based on Figures 9-8, 11-24, and 11-27, can you complete the design of the data management layer for Brian? (*Hint:* For some help, look at Figure 11-22.)

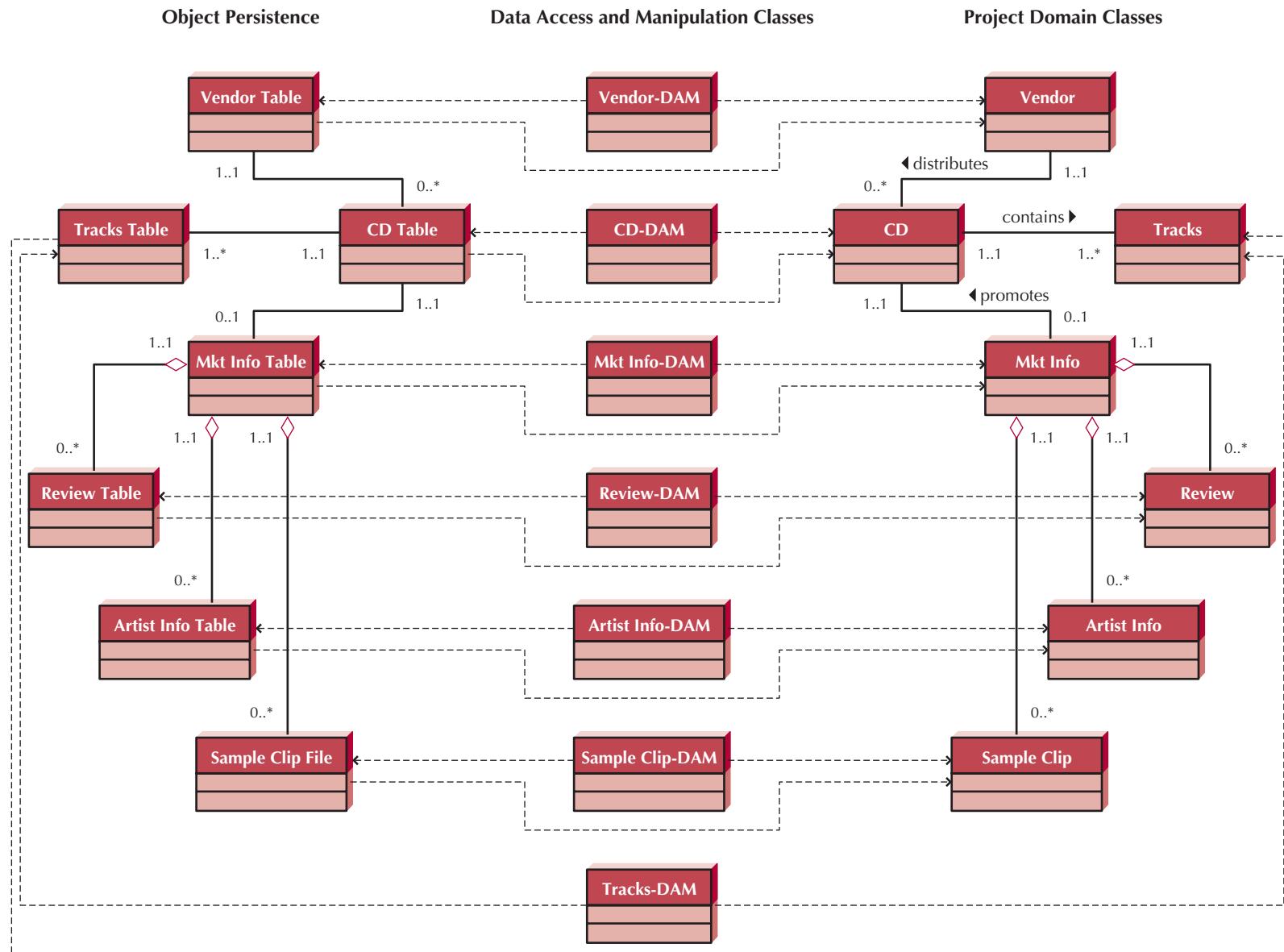


FIGURE 11-27 Data Management Layer and Problem Domain Layer Design for the CD Package of the Internet Sales System

SUMMARY

Object-Persistence Formats

There are four basic types of object-persistence formats: files (sequential and random access), object-oriented databases, object-relational databases, and relational databases. Files are electronic lists of data that have been optimized to perform a particular transaction. There are two different access methods (sequential and random), and there are five different application types: master, look-up, transaction, audit, and history. Master files typically are kept for long periods of time because they store important business information, such as order information or customer mailing information. Look-up files contain static values that are used to validate fields in the master files, and transaction files temporarily hold information that will be used for a master file update. An audit file records before and after images of data as they are altered so that an audit can be performed if the integrity of the data is questioned. Finally, the history file stores past transactions (e.g., old customers, past orders) that are no longer needed by the system.

A database is a collection of information groupings that are related to each other in some way, and a DBMS (database management system) is software that creates and manipulates databases. There are three types of databases that are likely to be encountered during a project: relational, object-relational, and object-oriented. The relational database is the most popular kind of database for application development today. It is based on collections of tables that are related to each other through common fields, known as foreign keys. Object-relational databases are relational databases that have extensions that provide limited support for object-orientation. The extensions typically include some support for the storage of objects in the relational table structure. Object-oriented databases come in two flavors: full-blown DBMS products and extensions to an object-oriented programming language. Both approaches typically fully support object-orientation.

Selecting an Object-Persistence Format

The application's data should drive the storage format decision. Relational databases support simple data types very effectively; whereas, object databases are best for complex data. The type of system also should be considered when choosing among data storage formats (e.g., relational databases have matured to support transactional systems). Although less critical to the format selection decision, the project team needs to consider what technology exists within the organization and the kind of technology likely to be used in the future.

Mapping Problem Domain Objects to Object-Persistence Formats

There are many different approaches to support object persistence. Each of the different formats for object persistence has some conversion requirements. The complexity of the conversion requirements increases the further the format is from an object-oriented format. An OODBMS has the fewest conversion requirements while a RDBMS tends to have the most. No matter what format is chosen, all data management functionality should be kept out of the problem domain classes to minimize the maintenance requirements of the system and to maximize the portability and reusability of the problem domain classes.

Optimizing RDBMS-Based Object Storage

There are two primary dimensions by which to optimize a relational database: for storage efficiency and for speed of access. The most efficient relational database tables in terms of data storage are those that have no redundant data and very few null values. Normalization is the process whereby a series of rules are applied to the data management layer to

determine how well-formed it is. A model is in first normal form (1NF) if it does not lead to repeating fields, which are fields that repeat within a table to capture multiple values. Second normal form (2NF) requires that all tables be in 1NF and lead to fields whose values are dependent on the whole primary key. Third normal form (3NF) occurs when a model is in both 1NF and 2NF, and none of the resulting fields in the tables are dependent on non-primary key fields (i.e., transitive dependency). With each violation, additional tables should be created to remove the repeating fields or improper dependencies from the existing tables.

Once you have optimized the design of the object persistence for storage efficiency, the data may be spread out across a number of tables. To improve speed, the project team may decide to denormalize—add redundancy back into—the design. Denormalization reduces the number of joins that need to be performed in a query, thus speeding up data access. Denormalization is best in situations where data are accessed frequently and updated rarely. There are three modeling situations that are good candidates for denormalization: lookup tables, entities that share one-to-one relationships, and entities that share one-to-many relationships. In all three cases, attributes from one entity are moved or repeated in another entity to reduce the joins that must occur while accessing the database.

Clustering occurs when similar records are stored close together on the storage medium to speed up retrieval. In intra-file clustering, similar records in the table are stored together in some way, such as in sequence. Inter-file clustering combines records from more than one table that typically are retrieved together. Indexes also can be created to improve the access speed of a system. An index is a mini-table that contains values from one or more columns in a table and where the values can be found. Instead of performing a table scan, which is the most inefficient way to retrieve data from a table, an index points directly to the records that match the requirements of a query.

Finally, the speed of the system can be improved if the right hardware is purchased to support it. Analysts can use volumetrics to estimate the current and future size of the database and then share these numbers with the people who are responsible for buying and configuring the database hardware.

Designing Data Access and Manipulation Classes

Once the object persistence has been designed, a translation layer between the problem domain classes and the object persistence should be created. The translation layer is implemented through data access and manipulation classes. In this manner, any changes to the object-persistence format chosen will only require changes to the data access and manipulation classes. The problem domain classes will be completely isolated from the changes.

KEY TERMS

Attribute sets	Denormalization	Foreign key
Audit file	End-user DBMS	History file
Clustering	Enterprise DBMS	Impedance mismatch
Data access and manipulation classes	Executive information systems (EIS)	Index
Data management layer	Expert system (ES)	Inter-file cluster
Database	Extent	Intra-file cluster
Database management system (DBMS)	File	Join
Decision support systems (DSS)	First normal form (1NF)	Linked list

Look-up file	ment system (ORDBMS)	Relationship sets
Management information system (MIS)	Ordered sequential access file	Repeating groups (fields)
Master file	Overhead	Second normal form (2NF)
Multi-valued attributes (fields)	Partial dependency	Sequential access files
Normalization	Pointer	Structured query language (SQL)
Object ID	Primary key	Table scan
Object-oriented database management system (OODBMS)	Problem domain classes	Third normal form (3NF)
Object-oriented programming language (OOPL)	Random access files	Transaction file
Object persistence formats	Raw data	Transaction processing system
Object-relational database management system (RDBMS)	Referential integrity	Transitive dependency
	Relational database	Unordered sequential access file
	base management	Update anomaly
	system (RDBMS)	Volumetrics

QUESTIONS

1. Describe the four steps to object persistence design.
2. How are a file and a database different from each other?
3. What is the difference between an end-user database and an enterprise database? Provide an example of each one.
4. What are the differences between sequential and random access files?
5. Name five types of application files and describe the primary purpose of each type.
6. What is the most popular kind of database today? Provide three examples of products that are based on this database technology.
7. What is referential integrity and how is it implemented in a RDBMS?
8. List some of the differences between an ORDBMS and a RDBMS.
9. What are the advantages of using an ORDBMS over a RDBMS?
10. List some of the differences between an ORDBMS and an OODBMS.
11. What are the advantages of using an ORDBMS over an OODBMS?
12. What are the advantages of using an OODBMS over a RDBMS?
13. What are the advantages of using an OODBMS over an ORDBMS?
14. What are the factors in determining the type of object-persistence format that should be adopted for a system? Why are these factors so important?
15. Why should you consider the storage formats that already exist in an organization when deciding upon a storage format for a new system?
16. When implementing the object persistence in an ORDBMS, what types of issues must you address?
17. When implementing the object persistence in an RDBMS, what types of issues must you address?
18. Name three ways in which null values can be interpreted in a relational database. Why is this problematic?
19. What are the two dimensions in which to optimize a relational database?
20. What is the purpose of normalization?
21. How does a model meet the requirements of third normal form?
22. Describe three situations that can be good candidates for denormalization.
23. Describe several techniques that can improve performance of a database.
24. What is the difference between inter-file and intra-file clustering? Why are they used?
25. What is an index and how can it improve the performance of a system?
26. Describe what should be considered when estimating the size of a database.
27. Why is it important to understand the initial and projected size of a database during the design phase?
28. What are the key issues in deciding between using perfectly normalized databases and denormalized databases?
29. What is the primary purpose of the data access and manipulation classes?
30. Why should the data access and manipulation classes be dependent on the associated problem domain classes instead of the other way around?
31. Why should the object persistence be dependent on the associated problem domain classes instead of the other way around?
32. Why should the data access and manipulation classes be dependent on the associated object persistence instead of the other way around?

EXERCISES

- A. Using the Web or other resources, identify a product that can be classified as an end-user database and a product that can be classified as an enterprise database. How are the products described and marketed? What kinds of applications and users do they support? In what kinds of situations would an organization choose to implement an end-user database over an enterprise database?
- B. Visit a commercial Web site (e.g., Amazon.com). If files were being used to store the data supporting the application, what types of files would be needed? What access type would be required? What data would they contain?
- C. Using the Web, review one of the products listed below. What are the main features and functions of the software? In what companies has the DBMS been implemented, and for what purposes? According to the information that you found, what are three strengths and weaknesses of the product?
1. Relational DBMS
 2. Object-relational DBMS
 3. Object-oriented DBMS
- D. You have been given a file that contains the following fields relating to CD information. Using the steps of normalization, create a model that represents this file in third normal form. The fields include:
- | | |
|-----------------------|-------------|
| Musical group name | CD title 2 |
| Musicians in group | CD title 3 |
| Date group was formed | CD 1 length |
| Group's agent | CD 2 length |
| CD title 1 | CD 3 length |
- Assumptions:
- Musicians in group contains a list of the members of the people in the musical group.
- E. Musical groups can have more than one CD, so both group name and CD title are needed to uniquely identify a particular CD.
- F. Jim Smith's dealership sells Fords, Hondas, and Toyotas. The dealership keeps information about each car manufacturer with whom they deal so that they can get in touch with them easily. The dealership also keeps information about the models of cars that they carry from each manufacturer. They keep information like list price, the price the dealership paid to obtain the model, and the model name and series (e.g., Honda Civic LX). They also keep information about all sales that they have made (for instance, they will record the buyer's name, the car they bought, and the amount they paid for the car). In order to contact the buyers in the future, contact information is also kept (e.g., address, phone number). Create a class diagram for this situation. Apply the rules of normalization to the class diagram to "check" the diagram for processing efficiency.
- G. Describe how you would denormalize the model that you created in Question E. Draw the new class diagram based on your suggested changes. How would performance be affected by your suggestions?
- H. Examine the model that you created in Question F. Develop a clustering and indexing strategy for this model. Describe how your strategy will improve the performance of the database.
- I. Calculate the size of the database that you created in Question F. Provide size estimates for the initial size of the database as well as for the database in one year's time. Assume that the dealership sells ten models of cars from each manufacturer to approximately 20,000 customers a year. The system will be set up initially with one year's worth of data.

MINICASES

1. The system development team at the Wilcon Company is working on developing a new customer order entry system. In the process of designing the new system, the team has identified the following class and its attributes:

Inventory Order
Order Number (PK)
Order Date
Customer Name
Street Address
City
State
Zip
Customer Type
Initials
District Number
Region Number
1 to 22 occurrences of:
Item Name
Quantity Ordered
Item Unit
Quantity Shipped
Item Out
Quantity Received

- State the rule that is applied to place a class in first normal form. Revise the above class diagram so that it is in first normal form.
- State the rule that is applied to place a class into second normal form. Create a class diagram for the Wilcon Company using the class and attributes described (if necessary) to place it in second normal form.

- State the rule that is applied to place a class into third normal form. Revise the class diagram to place it in third normal form.

- When planning for the physical design of this database, can you identify any likely situations where the project team may choose to denormalize the class diagram? After going through the work of normalizing, why would this be considered?

- In the new system under development for Holiday Travel Vehicles, seven tables will be implemented in the new relational database. These tables are: New Vehicle, Trade-in Vehicle, Sales Invoice, Customer, Salesperson, Installed Option, and Option. The expected average record size for these tables and the initial record count per table are given below.

Table Name	Average Record Size	Initial Table Size (records)
New Vehicle	65 characters	10,000
Trade-in Vehicle	48 characters	7,500
Sales Invoice	76 characters	16,000
Customer	61 characters	13,000
Salesperson	34 characters	100
Installed Option	16 characters	25,000
Option	28 characters	500

Perform a volumetrics analysis for the Holiday Travel Vehicle system. Assume that the DBMS that will be used to implement the system requires 35 percent overhead to be factored into the estimates. Also, assume a growth rate for the company of 10 percent per year. The systems development team wants to ensure that adequate hardware is obtained for the next three years.

CHAPTER 12

HUMAN COMPUTER INTERACTION LAYER DESIGN

A user interface is the part of the system with which the users interact. It includes the screen displays that provide navigation through the system, the screens and forms that capture data, and the reports that the system produces (whether on paper, on the screen, or via some other media). This chapter introduces the basic principles and processes of interface design, discusses how to design the interface structure and standards, navigation design, input design, and output design.

OBJECTIVES:

- Understand several fundamental user interface design principles.
- Understand the process of user interface design.
- Understand how to design the user interface structure.
- Understand how to design the user interface standards.
- Understand commonly used principles and techniques for navigation design.
- Understand commonly used principles and techniques for input design.
- Understand commonly used principles and techniques for output design.
- Be able to design a user interface.

CHAPTER OUTLINE

Introduction	Navigation Design Documentation
Principles for User Interface Design	Input Design
Layout	Basic Principles
Content Awareness	Types of Inputs
Aesthetics	Input Validation
User Experience	Output Design
Consistency	Basic Principles
Minimize User Effort	Types of Outputs
User Interface Design Process	Media
Use Scenario Development	Applying the Concepts at CD Selections
Interface Structure Design	Use Scenario Development
Interface Standards Design	Interface Structure Design
Interface Design Prototyping	Interface Standards Design
Interface Evaluation	Interface Template Design
Navigation Design	Design Prototyping
Basic Principles	Interface Evaluation
Types of Navigation Controls	Summary
Messages	

INTRODUCTION

Interface design is the process of defining how the system will interact with external entities (e.g., customers, suppliers, other systems). In this chapter we focus on the design of *user interfaces*, but it is also important to remember that there are sometimes *system interfaces* that exchange information with other systems. System interfaces are typically designed as part of a systems integration effort. They are defined in general terms as part of the physical architecture and data management layers.

The human computer interface layer defines the way in which the users will interact with the system and the nature of the inputs and outputs that the system accepts and produces. The user interface includes three fundamental parts. The first is the *navigation mechanism*, the way in which the user gives instructions to the system and tells it what to do (e.g., buttons, menus). The second is the *input mechanism*, the way in which the system captures information (e.g., forms for adding new customers). The third is the *output mechanism*, the way in which the system provides information to the user or to other systems (e.g., reports, Web pages). Each of these is conceptually different, but they are closely intertwined. All computer displays contain navigation mechanisms, and most contain input and output mechanisms. Therefore, navigation design, input design, and output design are tightly coupled.

First, this chapter introduces several fundamental user-interface design principles. Second, it provides an overview of the design process for the human computer interaction layer. Third, the chapter provides an overview of the navigation, input and output components that are used in interface design. This chapter focuses on the design of Web-based interfaces and *Graphical User Interfaces (GUI)* that use windows, menus, icons, and a mouse (e.g., Windows, Macintosh).¹ Although text-based interfaces are still commonly used on mainframes and Unix systems, GUI interfaces are probably the most common type of interfaces that you will use, with the possible exception of printed reports.²

PRINCIPLES FOR USER INTERFACE DESIGN

In many ways, user interface design is an art. The goal is to make the interface pleasing to the eye and simple to use, while minimizing the effort the users need to accomplish their work. The system is never an end in itself; it is merely a means to accomplish the “business” of the organization.

We have found that the greatest problem facing experienced designers is using space effectively. Simply put, there often is too much information that needs to be presented on a screen or report or form than will fit comfortably. Analysts must balance the need for simplicity and pleasant appearance against the need to present the information across multiple pages or screens, which decreases simplicity. In this section, we discuss some fundamental interface design principles, which are common for navigation design, input design, and output design³ (see Figure 12-1).

¹ Many people attribute the origin of GUI interfaces to Apple or Microsoft. Some people know that Microsoft copied from Apple, which in turn “borrowed” the whole idea from a system developed at the Xerox Palo Alto Research Center (PARC) in the 1970s. Very few know that the Xerox system was based on a system developed by Doug Englebart of Stanford that was first demonstrated at the Western Computer Conference in 1968. Around the same time, Doug also invented the mouse, desktop video conferencing, groupware, and host of other things we now take for granted. Doug is a legend in the computer science community and has won too many awards to count, but is relatively unknown by the general public.

² A good book on GUI design is Susan Fowler, *GUI Design Handbook* (New York, McGraw-Hill, 1998).

³ A good book on the design of interfaces is Susan Weinschenk, Pamela Jamar, and Sarah Yeo, *GUI Design Essentials* (New York, Wiley, 1997).

Principle	Description
Layout	The interface should be a series of areas on the screen that are used consistently for different purposes—for example, a top area for commands and navigation, a middle area for information to be input or output, and a bottom area for status information.
Content awareness	Users should always be aware of where they are in the system and what information is being displayed.
Aesthetics	Interfaces should be functional and inviting to users through careful use of white space, colors, and fonts. There is often a tradeoff between including enough white space to make the interface look pleasing without losing so much space that important information does not fit on the screen.
User experience	Although ease of use and ease of learning often lead to similar design decisions, there is sometimes a tradeoff between the two. Novice users or infrequent users of software will prefer ease of learning, whereas frequent users will prefer ease of use.
Consistency	Consistency in interface design enables users to predict what will happen before they perform a function. It is one of the most important elements in ease of learning, ease of use, and aesthetics.
Minimal user effort	The interface should be simple to use. Most designers plan on having no more than three mouse clicks from the starting menu until users perform work.

FIGURE 12-1

Principles of User Interface Design

Layout

The first element of design is the basic *layout* of the *screen*, *form*, or *report*. Most software designed for personal computers follows the standard Windows or Macintosh approach for screen design. The screen is divided into three boxes. The top box is the navigation area through which the user issues commands to navigate through the system. The bottom box is the status area, which displays information about what the user is doing. The middle—and largest—box is used to display reports and present forms for data entry.

In many cases (particularly on the Web), multiple layout areas are used. Figure 12-2 shows a screen with five navigation areas, each of which is organized to provide different functions and navigation within different parts of the system. The top area provides the standard Internet Explorer navigation and command controls that change the contents of the entire system. The navigation area on the left edge navigates between sections and changes all content to its right. The other two section navigation areas at the top and bottom of the page provide other ways to navigate between sections. The content in the middle of the page displays the results (i.e., a report on a book) plus provides additional navigation within the page about this book.

This use of multiple layout areas for navigation also applies to inputs and outputs. Data areas on reports and forms are often subdivided into subareas, each of which is used for different types of information. These areas are almost always rectangular in shape, although sometimes space constraints will require odd shapes. Nonetheless, the margins on the edges of the screen should be consistent. Each of the areas within the report or form is designed to hold different information. For example, on an order form (or order report) one part may be used for customer information (e.g., name, address), one part for information about the order in general (e.g., date, payment information), and one part for the order details (e.g., how many units of which items at what price each). Each area is self-contained so that information in one area does not run into another.

The areas and information within areas should have a natural intuitive flow to minimize the users' movement from one area to the next. People in westernized nations (e.g., United

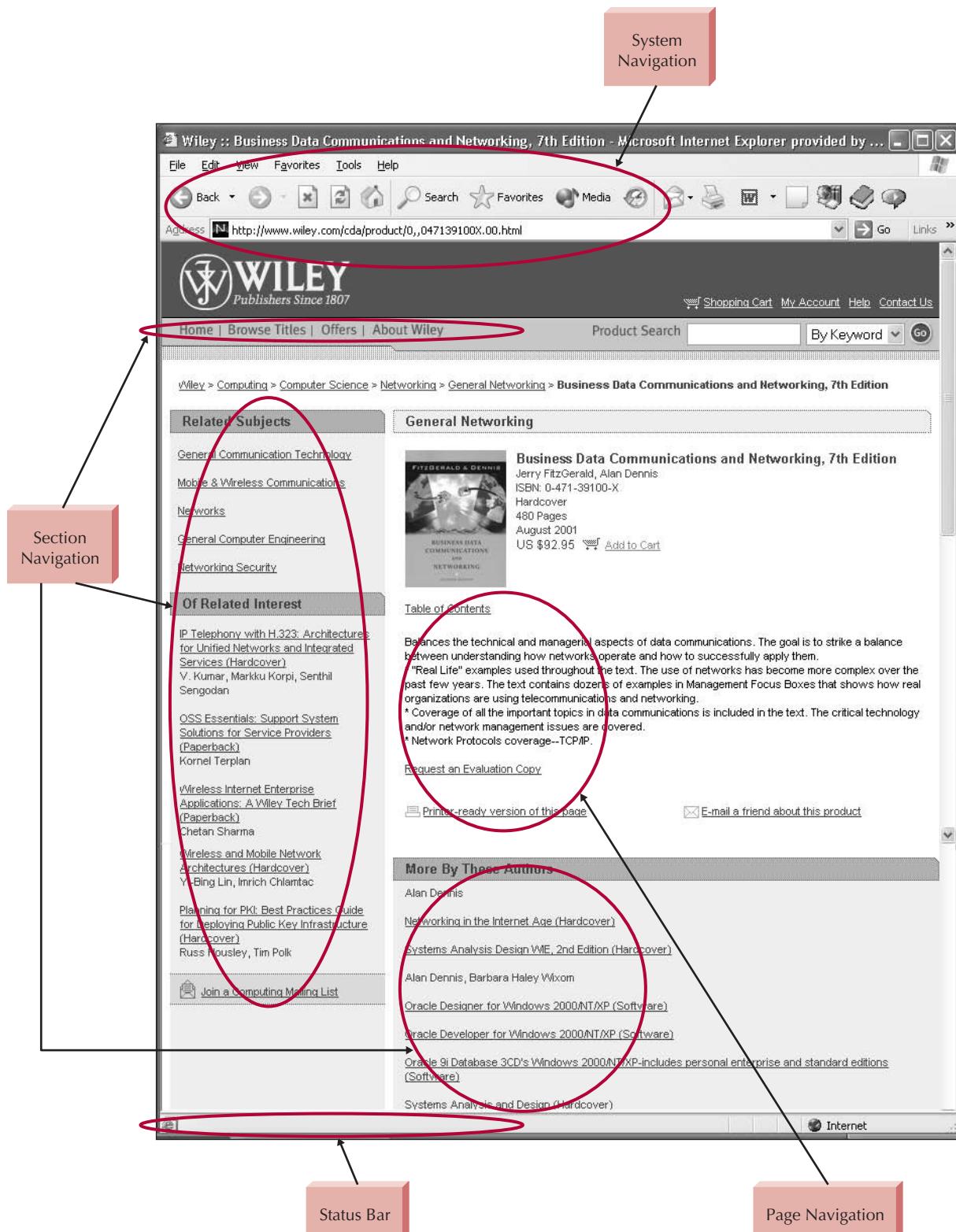


FIGURE 12-2 Layout with Multiple Navigation Areas

States, Canada, Mexico) tend to read top-to-bottom left-to-right, so related information should be placed so it is used in this order (e.g., address lines, followed by city, state/province, and then zip code/postal code). Sometimes the sequence is in chronological order, or from the general to the specific, or from most frequently to least frequently used. In any event, before the areas are placed on a form or report, the analyst should have a clear understanding of what arrangement makes the most sense for how the form or report will be used. The flow between sections should also be consistent, whether horizontal or vertical (see Figure 12-3). Ideally, the areas will remain consistent in size, shape, and placement for the forms used to enter information (whether paper or on-screen) and the reports used to present it.

Content Awareness

Content awareness refers to the ability of an interface to make the user aware of the information it contains with the least amount of effort on the user's part. All parts of the interface, whether navigation, input or output, should provide as much content awareness as possible, but it is particularly important for forms or reports that are used quickly or irregularly (e.g., a Web site).

Content awareness applies to the interface in general. All interfaces should have titles (on the screen frame, for example). Menus should show where you are and, if possible, where you came from to get there. For example, in Figure 12-2, the line below the top section navigation area shows that the user has browsed from the Wiley Home page, to the Computer Science section, to the Networking section, to General Networking section, and then to this specific book.

Content awareness also applies to the areas within forms and reports. All areas should be clear and well defined (with titles if space permits) so that it is difficult for the user to become confused about the information in any area. Then users can quickly locate the part of the form or report that is likely to contain the information they need. Sometimes the areas are marked using lines, colors, or headings (e.g., the section area in Figure 12-2), while in other cases the areas are only implied (e.g., the page controls at the bottom of Figure 12-2).

Content awareness also applies to the *fields* within each area. Fields are the individual elements of data that are input or output. The *field labels* that identify the fields on the interface should be short and specific—objectives that often conflict. There should be no uncertainty about the format of information within fields, whether for entry or display. For example a date of 10/5/03 is different depending on whether you are in the United States (October 5, 2003) or in Canada (May 10, 2003). Any fields for which there is the possibility of uncertainty or multiple interpretations should provide explicit explanations.

Content awareness also applies to the information that a form or report contains. In general, all forms and reports should contain a preparation date (i.e., the date printed or the data completed) so that the age of information is obvious. Likewise, all printed forms and software should provide version numbers so that users, analysts, and programmers can identify outdated materials.

Figure 12-4 shows a form from the University of Georgia. This form illustrates the logical grouping of fields into areas with an explicit box (top left), as well as an implied area with no box (lower left). The address fields within the address area follow a clear, natural order. Field labels are short where possible (see the top left) but long where more information is needed to prevent misinterpretation (see the bottom left).

Aesthetics

Aesthetics refers to designing interfaces that are pleasing to the eye. Interfaces do not have to be works of art, but they do need to be functional and inviting to use. In most cases, “less is more,” meaning that a simple, minimalist design is the best.

Patient Information

Patient Name:

First Name:

Last Name:

Address:

Street:

City:

State/Province:

Zip Code/Postal Code:

Home phone:

Office phone:

Cell phone:

Referring Doctor:

First Name:

Last Name:

Street:

City:

State/Province:

Zip Code/Postal Code:

Office phone:

(A) Vertical Flow

Patient Information

Patient Name:

First Name: <input type="text"/>	Last Name: <input type="text"/>
----------------------------------	---------------------------------

Street: City: State/Province: Zip Code/Postal Code:

Home Phone: Office Phone: Cell Phone:

Referring Doctor:

First Name: <input type="text"/>	Last Name: <input type="text"/>
----------------------------------	---------------------------------

Street: City: State/Province: Zip Code/Postal Code:

Office Phone:

(B) Horizontal Flow

FIGURE 12-3 Flow Between Interface Sections (A) Vertical (B) Horizontal

EMPLOYEE PERSONNEL REPORT										UNIVERSITY OF GEORGIA																			
DOCUMENT NO.	PAGE	DATE	FY	DEPARTMENT	PHONE	COLLEGE OR DIVISION				UGA EMPLOYMENT HISTORY																			
DEPARTMENT/PROJECT					PRI. DEPT	HIGH DEGREE	INSTITUTION			YEAR	(C) CURRENT					(P) PREVIOUS													
SOC. SEC. NUM.	LAST NAME				FIRST NAME/INITIAL			MIDDLE INITIAL/NAME			SUF	DATE					PAY TYPE												
STREET OR ROUTE NO. (LINE 1)					NON-WORK PHONE		BIRTH DATE		SPOUSE'S NAME		CHAIR	UGA % TIME					ACTION MO DA YR												
STREET OR ROUTE NO. (LINE 2)					UNIVERSITY PHONE		CITIZEN OF		I-9 VISA		COUNTY	(1) REGULAR					(3) TEMPORARY												
CITY					STATE	ZIP + 4	UNIVERSITY BUILDING NAME			BLDG. NO./FLOOR/ROOM			(2) UGA STUDENT					(4) NR-ALIEN											
FOR PAYROLL DEPT USE ONLY										COOP. EXT. EMPLOYEES ONLY PAYROLL PAYMENT DISTRIBUTION																			
FED EXM	STATE EXM	OASDI	RETIRE	GDCP	COUNTY MONEY (PER PAY PERIOD)					UGA SALARY					(T) TIPPED														
		HI	EIC												(1) SEND TO DEPT (DIST CODE)														
TRX HOME SHORT POSN APPT. BEGIN MO DA YR HR APPT. END MO DA YR HR JOBCLASS POSITION TITLE										COOP. EXT. EMPLOYEES ONLY PAYROLL PAYMENT DISTRIBUTION																			
DEPT TITLE NO.										TOTAL										(2) DIRECT DEPOSIT(SEND PR105 TO PAYROLL)									
DEPT TITLE NO.										TOTAL										(3) PICK UP AT PAYROLL WINDOW									
PAYROLL AUTHORIZATION										FISCAL YEAR																			
TRX	HOME	SHORT	POSN	ACCOUNT	EFT	BUDGET	FROM	MO	DA	YR	HR	MO	DA	YR	HR	MO	DA	YR	HR	MO	DA	YR	HR						
							THRU																						
							AMOUNT																						
							PER																						
							PAY																						
							PERIOD																						
							OR																						
							HOURLY																						
							RATE																						
TOTALS																													
<input type="checkbox"/> (A) NEW UGA EMPLOYEE <input type="checkbox"/> (B) LATERAL TRANSFER <input type="checkbox"/> (C) PROMOTION <input type="checkbox"/> (D) REPLACEMENT POSN-NAME OF LAST INCUMBENT _____ <input type="checkbox"/> (E) APPOINTMENT TO NEW POSITION <input type="checkbox"/> (F) CHANGE % TIME EMPLOYED FROM _____ TO _____ <input type="checkbox"/> (G) CONTINUATION WITHIN EXISTING BUDGET POSITION <input type="checkbox"/> (H) REVISE DISTRIBUTION OF SALARY <input type="checkbox"/> (I) TRANSFER FROM DEPT _____ TO _____ <input type="checkbox"/> (J) CHANGE PAY TYPE FROM _____ TO _____										<input type="checkbox"/> (K) CHANGE TITLE FROM _____ TO _____ <input type="checkbox"/> (L) CHANGE NAME FROM _____ TO _____ <input type="checkbox"/> (M) CHANGE SSN FROM _____ TO _____ <input type="checkbox"/> (N) LEAVE W/O PAY FROM _____ TO _____ <input type="checkbox"/> (O) CHG COUNTY \$ FROM _____ TO _____ <input type="checkbox"/> (P) TERMINATION-REASON _____ <input type="checkbox"/> (Q) OTHER (SPECIFY) _____																			
REMARKS																													
DEPARTMENT HEAD				DATE		VICE PRESIDENT				DATE		BUDGET REVIEW				DATE		BUDGET OFFICE				DATE							
DEAN/DIRECTOR				DATE		FACULTY RECORDS				DATE		CONTRACTS & GRANTS				DATE		PERSONNEL				DATE							

FIGURE 12-4 Form Example

Space is usually at a premium on forms and reports, and often there is the temptation to squeeze as much information as possible onto a page or a screen. Unfortunately, this can make a form or report so unpleasant that users do not want to use it. In general, all forms and reports need a minimum amount of *white space* that is intentionally left blank.

What was your first reaction when you looked at Figure 12-4? This is the most unpleasant form at University of Georgia, according to staff members. Its *density* is too high; it has too much information packed into too small of a space with too little white space. While it may be efficient in saving paper by using one page not two, it is not effective for many users.

In general, novice or infrequent users of an interface, whether on a screen or on paper, prefer interfaces with low density, often one with a density of less than 50 percent (i.e., less than 50 percent of the interface occupied by information). More experienced users prefer higher densities, sometimes approaching 90 percent occupied because they know where information is located and high densities reduce the amount of physical movement through the interface. We suspect the form in Figure 12-4 was designed for the experienced staff in the personnel office who use it daily, rather than for the clerical staff in academic departments with less personnel experience who use the form only a few times a year.

The design of text is equally important. As a general rule, all text should be in the same font and about the same size. Fonts should be no less than 8 points in size, but 10 points is often preferred, particularly if the interface will be used by older people. Changes in font and size are used to indicate changes in the type of information that is presented (e.g., headings, status indicators). In general, italics and underlining should be avoided because they make text harder to read.

Serif fonts (i.e., those having letters with serifs or tails [e.g., Times Roman], such as the font you are reading right now) are the most readable for printed reports, particularly for small letters. San-serif fonts (i.e., those without serifs [e.g., Helvetica or Arial] such as the ones used for the chapter titles in this book) are the most readable for computer screens and are often used for headings in printed reports. Never use all capital letters, except possibly for titles.

Color and patterns should be used carefully and sparingly and only when they serve a purpose. (About 10 percent of men are color blind, so the improper use of color can impair their ability to read information.) A quick trip around the Web will demonstrate the problems caused by indiscriminate use of colors and patterns. Remember, the goal is pleasant readability, not art; color and patterns should be used to strengthen the message, not overwhelm it. Color is best used to separate and categorize items, such as showing the difference between headings and regular text, or to highlight important information. Therefore, colors with high contrast should be used (e.g., black and white). In general, black text on a white background is the most readable, with blue on red the least readable. (Most experts agree that background patterns on Web pages should be avoided.) Color has been shown to affect emotion, with red provoking intense emotion (e.g., anger) and blue provoking lowered emotions (e.g., drowsiness).

User Experience

User experience, essentially, can be broken down into two levels: those with experience, and those without. Interfaces should be designed for both types of users. Novice users usually are most concerned with *ease of learning*—how quickly they can learn new systems. Expert users are usually most concerned with *ease of use*—how quickly they can use the system once they have learned how to use it. Often these two are complementary and lead to similar design decisions, but sometimes there are trade-offs. Novices, for example, often prefer menus, which show all available system functions, because these promote ease of learning. Experts, on the other hand, sometimes prefer fewer menus that are organized around the most commonly used functions.

Systems that will end up being used by many people on a daily basis are more likely to have a majority of expert users (e.g., order entry systems). Although interfaces should try to balance ease of use and ease of learning, these types of systems should put more emphasis on ease of use, rather than ease of learning. Users should be able to access the commonly used functions quickly, with few keystrokes or a small number of menu selections.

In many other systems (e.g., decision support systems), most people will remain occasional users for the lifetime of the system. In this case, greater emphasis may be placed on ease of learning rather than ease of use.

Ease of use and ease of learning often go hand-in-hand—but sometimes they don't. Research shows that expert and novice users have different requirements and behavior patterns in some cases. For example, novices virtually never look at the bottom area of a screen that presents status information, while experts refer the status bar when they need information. Most systems should be designed to support frequent users, except for systems designed to be used infrequently or when many new users or occasional users are expected (e.g., the Web). Likewise, systems that contain functionality that is used only occasionally must contain a highly intuitive interface, or an interface that contains clear explicit guidance regarding its use.

The balance of quick access to commonly used and well-known functions, and guidance through new and less well known functions is challenging to the interface designer, and this balance often requires elegant solutions. Microsoft Office, for example, addresses this issue through the use of the “show me” functions that demonstrate the menus and buttons for specific functions. These features remain in the background until they are needed by novice users (or even experienced users when they use an unfamiliar part of the system).

Consistency

Consistency in design is probably the single most important factor in making a system simple to use because it enables users to predict what will happen. When interfaces are consistent, users can interact with one part of the system, and then know how to interact with the rest, aside of course, from elements unique to those parts. Consistency usually refers to the interface within one computer system, so that all parts of the same system work in the same way. Ideally, the system also should be consistent with other computer systems in the organization, and with commercial software that is used (e.g., Windows). For example, many users are familiar with the Web, so the use of Web-like interfaces can reduce the amount of learning required by the user. In this way, the user can reuse Web knowledge, thus significantly reducing the learning curve for a new system. Many software development tools support consistent system interfaces by providing standard interface objects (e.g., list boxes, pull down menus, and radio buttons).

Consistency occurs at many different levels. Consistency in the navigation controls conveys how actions in the system should be performed. For example, using the same icon or command to change an item clearly communicates how changes are made throughout the system. Consistency in terminology is also important. This refers to using the same words for elements on forms and reports (e.g., not “customer” in one place and “client” in another). We also believe that consistency in report and form design is important, although a recent study suggests that being too consistent can cause problems.⁴ When reports and forms are very similar except for very minor changes in titles, users sometimes mistakenly use the wrong form, and either enter incorrect data or misinterpret its information. The implication for design is to make the reports and forms similar, but give them some distinctive elements (e.g., color, size of titles) that enable users to immediately detect differences.

⁴ John Satzinger and Lorne Olfman, “User Interface Consistency Across End-User Application: The Effects of Mental Models,” *Journal of Management Information Systems* (Spring 1998), 167–193.

Minimize User Effort

Finally, interfaces should be designed to minimize the amount of effort needed to accomplish tasks. This means using the fewest possible mouse clicks or keystrokes to move from one part of the system to another. Most interface designers follow the *three clicks rule*: users should be able to go from the start or main menu of a system to the information or action they want in no more than three mouse clicks or three keystrokes.

USER INTERFACE DESIGN PROCESS

User interface design is a five-step process that is iterative—analysts often move back and forth between steps, rather than proceeding sequentially from step one to step five (see Figure 12-5). First, the analysts examine the *use cases* (see Chapter 6) and *sequence diagrams* (see Chapter 8) developed in the analysis phase and interview users to develop use scenarios that describe commonly employed patterns of actions the users will perform so the interface enables users to quickly and smoothly perform these scenarios. Second, the analysts develop a windows navigations system that defines the basic structure of the interface. These diagrams show all the interfaces (e.g., screens, forms, and reports) in the system and how they are connected. Third, the analysts design interface standards, which are the basic design elements on which interfaces in the system are based. Fourth, the analysts create an interface design prototype for each of the individual interfaces in the system, such as navigation controls (including the conversion of the *essential use cases* to *real use cases*), input screens, output screens, forms (including preprinted paper forms), and reports. Finally, the individual interfaces are subjected to interface evaluation to determine if they are satisfactory and how they can be improved.

Interface evaluations almost always identify improvements, so the interface design process is repeated in a cyclical process until no new improvements are identified. In practice, most analysts interact closely with the users during the interface design process so that users have many chances to see the interface as it evolves, rather than waiting for one overall interface evaluation at the end of the interface design process. It is better for all concerned (both analysts and users) if changes are identified sooner, rather than later. For example, if the interface structure or standards need improvements, it is better to identify changes before most of the screens that use the standards have been designed.⁵

Use Scenario Development

A *use scenario* is an outline of the steps that the users perform to accomplish some part of their work. A use scenario is one path through an essential use case. For example, Figure 6-17 shows the use case diagram for the Web section of the Internet sales system. This figure shows that the Maintain Order use case is distinct from the Checkout use case. We model the two use cases separately since they represent two separate processes that are included in the Place Order use case.

**YOUR
TURN**

12-1 Web Page Critique

Visit the Web home page for your university and navigate through several of its Web pages. Evaluate the extent to

which they meet the six design principles.

⁵ A good source for more information on user interface evaluation is Deborah Hix and H. Rex Hartson, *Developing User Interfaces: Ensuring Usability Through Product & Process* (New York: Wiley, 1993).

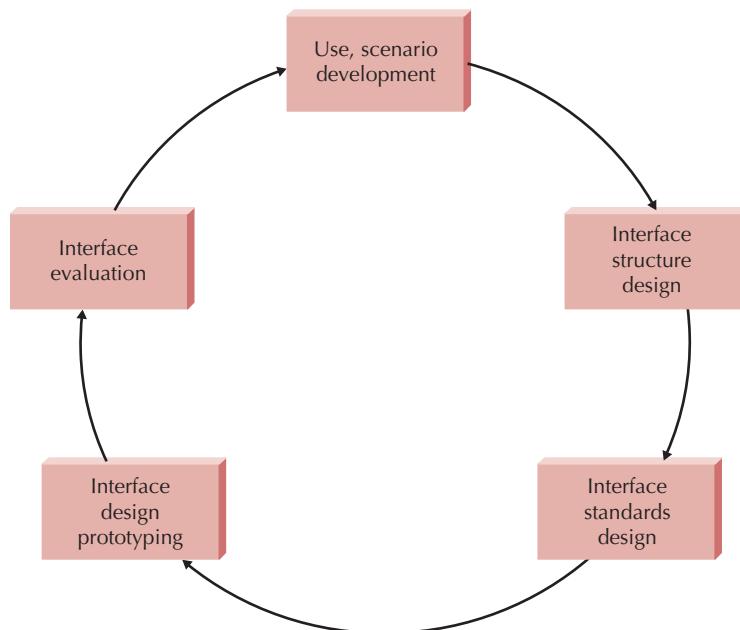


FIGURE 12-5
Use Interface Design
Process

The use case diagram was designed to model all possible uses of the system—its complete functionality or all possible paths through the use case at a fairly high-level of abstraction. In one use scenario, a user will browse for many CDs, much like someone browsing through a real music store looking for interesting CDs. He or she will search for a CD, read the marketing materials for it, perhaps add it to the shopping cart, browse for more, and so on. Eventually, the user will want to place the order, perhaps removing some things from the shopping cart beforehand.

In another use scenario, a user will want to buy one specific CD. He or she will go directly to the CD, price it, and buy it immediately, much like someone running into a store, making a beeline for the one CD he or she wants, and immediately paying and leaving the store. This user will enter the CD information in the search portion of the system, look at the resulting cost information, and immediately place an order. Anything that slows him or her down will risk loss of the sale. For this use scenario, we need to ensure that the path through the use case as presented by the interface is short and simple, with very few menus and mouse clicks.

Use scenarios are presented in a simple narrative description that is tied to the essential use cases developed during the analysis phase (see Chapter 6). Figure 12-6 shows the two use scenarios just described. The key point with using use cases for interface design is *not* to document all possible use scenarios within a use case. The goal is to document two or three of the *most common* use scenarios so the interface can be designed to enable the most common uses to be performed simply and easily.

**YOUR
TURN**

12-2 Use Scenario Development for the Web

Visit the Web home page for your university and navigate through several of its Web pages. Develop two use scenarios for it.

**YOUR
TURN**

12-3 Use Scenario Development for an ATM

Pretend you have been charged with the task of redesigning the interface for the ATM at your local bank. Develop two use scenarios for it.

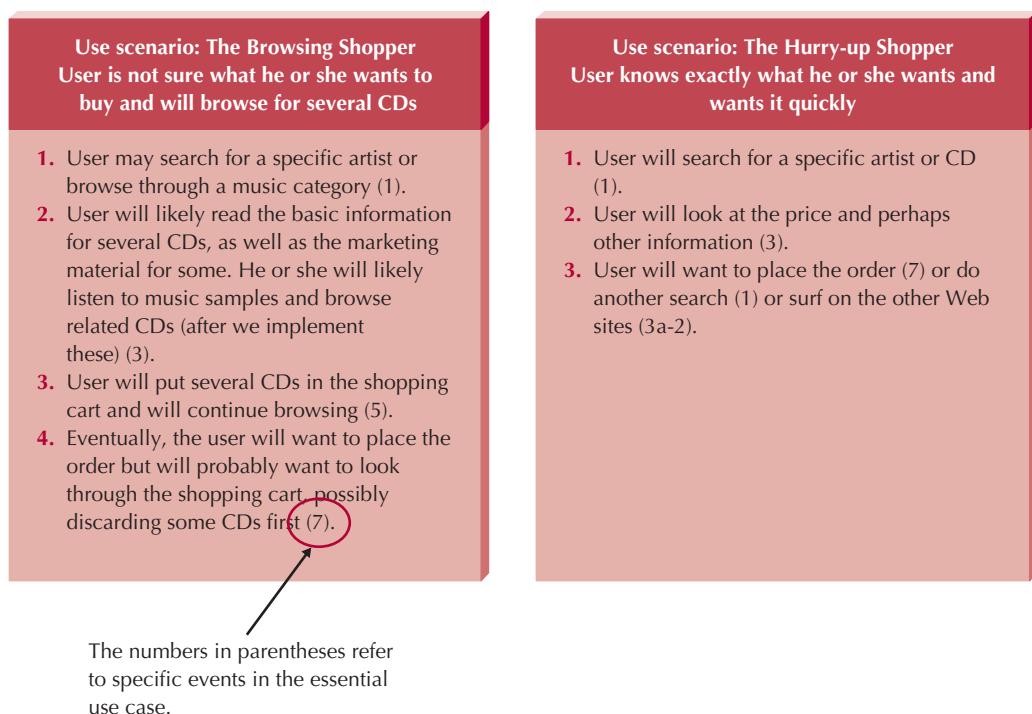


FIGURE 12-6 Use Scenarios

Interface Structure Design

The interface structure defines the basic components of the interface and how they work together to provide functionality to users. A *window navigation diagram* (*WND*)⁶ is used to show how all the screens, forms, and reports used by the system are related and how the user moves from one to another. Most systems have several WNDs, one for each major part of the system.

A WND is very similar to a behavioral state machine (see Chapter 8), in that they both model state changes. A behavioral state machine typically models the *state* changes of an object, whereas a WND models the state changes of the user interface. In a WND, each state that the user interface may be in is represented as a box. Furthermore, a box typically corresponds to a user interface component, such as a *window*, *form*, *button*, or *report*. For example in Figure 12-7, there are thirteen separate states: Main Menu, Menu A, Menu B, Menu C, and so on.

⁶ The window navigation diagram is actually an adaptation of the behavioral state machine and object diagrams (see Meilir Page-Jones, *Fundamentals of Object-Oriented Design in UML* [New York, NY: Dorset House, 2000]).

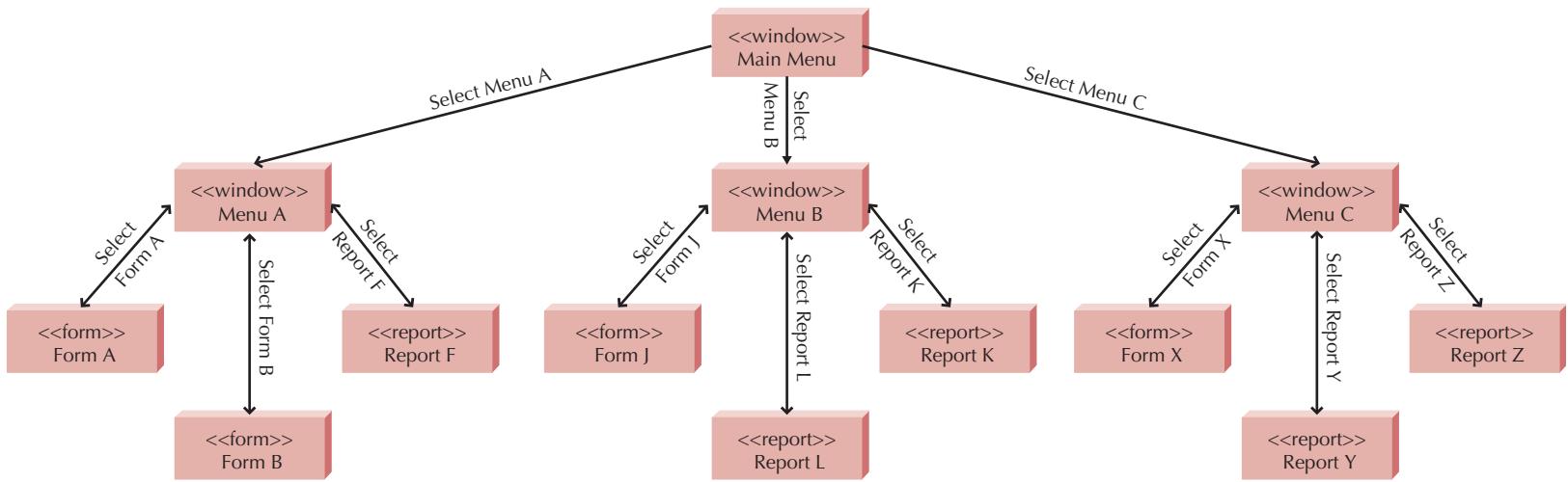


FIGURE 12-7 An Example Window Navigation Diagram

Transitions are modeled as either a single-headed or double-headed arrow. A single-headed arrow represents that a return to the calling state is not required, while a double-headed arrow represents a required return. For example in Figure 12-7, the transition from the Main Menu state to the Menu A state does not require a return; however, the transition from the Menu A state to the Form A state does. The arrows are labeled with the action that causes the user interface to move from one state to another. For example, in Figure 12-7, to move from the Main Menu state to the Menu A state, the user must “Select Menu A” from the Main Menu.

The last item to be described in a window navigation diagram is the *stereotype*. A stereotype is modeled as a text item enclosed within guillemets (<<>>) or angle brackets (<<>>). The stereotype represents the type of user interface component of a box on the diagram. For example, the Main Menu is a window, while Form A is a form.

The basic structure of the interface follows the basic structure of the business process itself as defined in the use cases and behavioral models. The analyst starts with the essential use cases and develops the fundamental flow of control of the system as it moves from object to object. The analyst then examines the use scenarios to see how well the WND supports them. Quite often, the use scenarios identify paths through the WND that are more complicated than they should be. The analyst then reworks the WND to simplify the ability of the interface to support the use scenarios, sometimes by making major changes to the menu structure, sometimes by adding shortcuts.

Interface Standards Design

The *interface standards* are the basic design elements that are common across the individual screens, forms, and reports within the system. Depending on the application, there may be several sets of interface standards for different parts of the system (e.g., one for Web screens, one for paper reports, one for input forms). For example, the part of the system used by data entry operators may mirror other data entry applications in the company, while a Web interface for displaying information from the same system may adhere to some standardized Web format. Likewise, each individual interface may not contain all of the elements in the standards (e.g., a report screen might not have an “edit” capability), and they may contain additional characteristics beyond the standard ones, but the standards serve as the touchstone that ensures the interfaces are consistent across the system. The following sections discuss some of the main areas in which interface standards should be considered: templates, metaphors, objects, actions, and icons.

Interface Templates The *interface template* defines the general appearance of all screens in the information system and the paper-based forms and reports that are used. The template design, for example, specifies the basic layout of the screens (e.g., where the navigation area(s), status area, and form/report area(s) will be placed) and the color scheme(s) that will be applied. It defines whether windows will replace one another on the screen or will cascade over the top of each other. The template defines a standard placement and order for common interface

YOUR
TURN

12-4 Interface Structure Design

Pretend you have been charged with the task of redesigning the interface for the ATM at your local bank. Design

an interface structure design using a WND that shows how a user would navigate among the screens.

actions (e.g., “File Edit View” rather than “File View Edit”). In short, the template draws together the other major interface design elements: metaphors, objects, actions, and icons.

Interface Metaphor First of all, the analysts must develop the fundamental *interface metaphor(s)* that defines how the interface will work. An interface metaphor is a concept from the real world that is used as a model for the computer system. The metaphor helps the user understand the system and enables the user to predict what features the interface might provide, even without actually using the system. Sometimes systems have one metaphor, while in other cases there are several metaphors in different parts of the system.

Often, the metaphor is explicit. Quicken, for example, uses a checkbook metaphor for its interface, even to the point of having the users type information into an on-screen form that looks like a real check. In other cases, the metaphor is implicit or unstated, but it is there, nonetheless. Many Windows systems use the paper form or table as a metaphor.

In some cases, the metaphor is so obvious that it requires no thought. The CD Selections Internet sales system, for example, will use the music store as the metaphor (e.g., shopping cart). In other cases, a metaphor is hard to identify. In general, it is better not to force a metaphor that really doesn’t fit a system, because an ill-fitting metaphor will confuse users by promoting incorrect assumptions.

Interface Objects The template specifies the names that the interface will use for the major *interface objects*, the fundamental building blocks of the system such as the classes. In many cases, the object names are straightforward, such as calling the shopping cart the “shopping cart.” In other cases, it is not so simple. For example, CD Selections sells both CDs and tapes. When users search for items to buy, should they search for “CDs” or “CDs & Tapes” or “CDs/Tapes” or “Music” or “Albums” or something else? Obviously, the object names should be easily understood and help promote the interface metaphor.

In general, in cases of disagreements between the users and the analysts over names, whether for objects or actions (discussed later), the users should win. A more “understandable” name always beats a more “precise” or more “accurate” one.

Interface Actions The template also specifies the navigation and command language style (e.g., menus), and grammar (e.g., object-action order; see the navigation design section later in this chapter). It gives names to the most commonly used *interface actions* in the navigation design (e.g., “buy” versus “purchase” or “modify” versus “change”).

Interface Icons The interface objects and actions and also their status (e.g., “deleted” or “overdrawn”) may be represented by *interface icons*. Icons are pictures that will appear on command buttons as well as in reports and forms to highlight important information. Icon design is very challenging because it means developing a simple picture less than half the size of a postage stamp that needs to convey an often-complex meaning. The simplest and best approach is to simply adopt icons developed by others (e.g., a blank page to indicate “create a new file,” a diskette to indicate “save”). This has the advantage of quick icon development, and the icons may already be well understood by users because they have seen them in other software.

Commands are actions that are especially difficult to represent with icons because they are in motion, not static. Many icons have become well known from widespread use, but icons are not as well understood as first believed. Use of icons can sometimes cause more confusion than insight. (For example, did you know that a picture of a sweeping broom [paintbrush?] in Microsoft Word means “format painter”?) Icon meanings become clearer with use, but sometimes a picture is not worth even one word; when in doubt, use a word not a picture.

**YOUR
TURN**

12-5 Interface Standards Development

Pretend you have been charged with the task of redesigning the interface for the ATM at your local bank. Develop

an interface standard that includes a template, metaphors, objects, actions, and icons.

Interface Design Prototyping

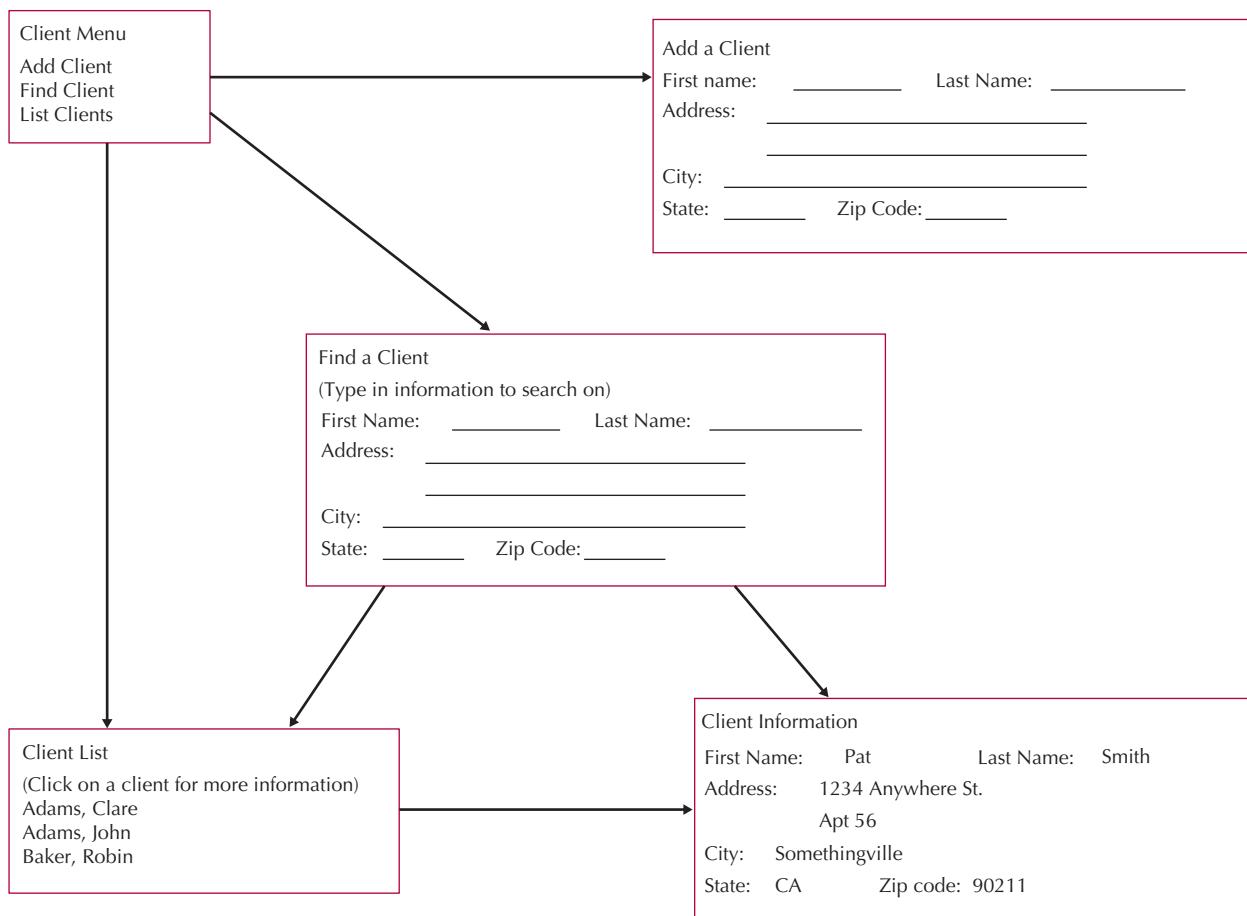
An *interface design prototype* is a mock-up or a simulation of a computer screen, form, or report. A prototype is prepared for each interface in the system to show the users and the programmers how the system will perform. In the “old days,” an interface design prototype was usually specified on a paper form that showed what would be displayed on each part of the screen. Paper forms are still used today, but more and more interface design prototypes are being built using computer tools instead of paper. The three most common approaches to interface design prototyping are storyboarding, HTML prototyping, and language prototyping.

Storyboard At its simplest, an interface design prototype is a paper-based *storyboard*. The storyboard shows hand-drawn pictures of what the screens will look like and how they flow from one screen to another, in the same way a storyboard for a cartoon shows how the action will flow from one scene to the next (see Figure 12-8). Storyboards are the simplest technique because all they require is paper (often a flip chart) and a pen—and someone with some artistic ability.

HTML Prototype One of the most common types of interface design prototypes used today is the *HTML prototype*. As the name suggests, an HTML prototype is built using Web pages created in HTML (hypertext mark-up language). The designer uses HTML to create a series of Web pages that shows the fundamental parts of the system. The users can interact with the pages by clicking on buttons and entering pretend data into forms (but because there is no “system” behind the pages the data are never processed). The pages are linked together so that as the user clicks on buttons, the requested part of the system appears. HTML prototypes are superior to storyboards in that they enable users to interact with the system and gain a better sense of how to navigate among the different screens. However, HTML has limitations—the screens shown in HTML will never appear exactly like the real screens in the system (unless, of course, the real system will be a Web system in HTML).

Language Prototype A *language prototype* is an interface design prototype built using the actual language or tool that will be used to build the system. Language prototypes are designed in the same ways as HTML prototypes (they enable the user to move from screen to screen, but perform no real processing). For example, in Visual Basic, it is possible to create and view screens without actually attaching program code to the screens. Language prototypes take longer to develop than storyboards or HTML prototypes, but have the distinct advantage of showing *exactly* what the screens will look like. The user does not have to guess about the shape or position of the elements on the screen.

Selecting the Appropriate Techniques Projects often use a combination of different interface design prototyping techniques for different parts of the system. Storyboarding is the fastest and least expensive, but provides the least amount of detail. Language

**FIGURE 12-8** An Example Storyboard

prototyping is the slowest, most expensive, and most detailed approach. HTML prototyping falls between the two extremes. Therefore storyboarding is used for parts of the system in which the interface is well understood and when more expensive prototypes are thought to be unnecessary. HTML prototypes and language prototypes are used for parts of the system that are critical yet not well understood.

Interface Evaluation

The objective of *interface evaluation* is to understand how to improve the interface design before the system is complete. Most interface designers intentionally or unintentionally design an interface that meets their personal preferences, which might or might not match the preferences of the users. The key message, therefore, is to have as many people as possible evaluate the interface, and the more users the better. Most experts recommend involving at least ten potential users in the evaluation process.

Many organizations save interface evaluation for the very last step in the systems development before the system is installed. Ideally, however, interface evaluation should be performed while the system is being designed—before it is built—so that any major design problems can be identified and corrected before the time and cost of programming has been spent on a weak design. It is not uncommon for the system to undergo one or two

CONCEPTS

IN ACTION

12-A Interface design prototypes for a DSS Application

I was involved in the development of several decision support systems (DSS) while working as a consultant. On one project, a future user was frustrated because he could not imagine what a DSS looked like and how one would be used. He was a key user, but the project team had a difficult time involving him in the project because of his frustration. The team used SQLWindows (one of the most popular development tools at the time) to create a language prototype that demonstrated the future system's appearance, proposed menu system, and screens (with fields, but no processing).

The team was amazed at the user's response to the prototype. He appreciated being given a context with which to visualize the DSS, and he soon began to recommend

improvements to the design and flow of the system, and to identify some important information that was overlooked during the analysis phase. Ultimately, the user became one of the strongest supporters of the system, and the project team felt sure that the prototype would lead to a much better product in the end.

—Barbara Wixom

Question:

1. Why do you think the team chose to use a language prototype rather than a storyboard or HTML prototype? What trade-offs were involved in the decision?

major changes after the users see the first interface design prototype because they identify problems that are overlooked by the project team.

As with interface design prototyping, interface evaluation can take many different forms, each with different costs and different amounts of detail. Four common approaches are heuristic evaluation, walk-through evaluation, interactive evaluation, and formal usability testing. As with interface design prototyping, the different parts of a system can be evaluated using different techniques.

Heuristic Evaluation A *heuristic evaluation* examines the interface by comparing it to a set of heuristics or principles for interface design. The project team develops a checklist of interface design principles—from the list at the start of this chapter, for example, as well as the list of principles in the navigation, input, and output design sections later in this chapter. At least three members of the project team then individually work through the interface design prototype examining each and every interface to ensure it satisfies each design principle on a formal checklist. After each has gone through the prototype separately, they meet as a team to discuss their evaluation and identify specific improvements that are required.

Walkthrough Evaluation An interface design is a meeting conducted with the users who will ultimately have to operate the system. The project team presents the prototype to the users and walks them through the various parts of the interface. The project team shows the storyboard or actually demonstrates the HTML or language prototype and explains how the interface will be used. The users identify improvements to each of the interfaces that are presented.

Interactive Evaluation With an *interactive evaluation*, the users themselves actually work with the HTML or language prototype in a one-person session with member(s) of the project team (an interactive evaluation cannot be used with a storyboard). As the user works with the prototype (often by going through the use scenarios, using the real use cases described later in this chapter, or just navigating at will through the system), he or she tells the project team member(s) what he or she likes and doesn't like, and what

additional information or functionality is needed. As the user interacts with the prototype, team member(s) record the cases when he or she appears to be unsure of what to do, makes mistakes, or misinterprets the meaning of an interface component. If the pattern of uncertainty, mistakes or misinterpretations reoccurs across several of the users participating in the evaluation, it is a clear indication that those parts of the interface need improvement.

Formal Usability Testing Formal *usability testing* is commonly done with commercial software products and products developed by large organizations that will be widely used through the organization. As the name suggests, it a very formal—almost scientific—process that can be used only with language prototypes (and systems that have been completely built awaiting installation or shipping).⁷ As with interactive evaluation, usability testing is done in one-person sessions in which a user works directly with the software. However, it is typically done in a special lab equipped with video cameras and special software that records each and every keystroke and mouse operation so they can be replayed to understand exactly what the user did.

The user is given a specific set of tasks to accomplish (usually the use scenarios) and after some initial instructions, the project teams member(s) are not permitted to interact with the user to provide assistance. The user must work with the software without help, which can be hard on the users if they become confused with the system. It is critical that users understand that the goal is to test the interface, not their abilities, and if they are unable to complete the task, the interface—not the user—has failed the test.

Formal usability testing is very expensive, because each one-user session can take one to two days to analyze due to the volume of detail collected in the computer logs and videotapes. Sessions typically last one to two hours. Most usability testing involves five to ten users, because fewer than five users makes the results depend too much on the specific individual users who participated, and more than ten users is often too expensive to justify (unless you work for a large commercial software developer).

NAVIGATION DESIGN

The navigation component of the interface enables the user to enter commands to navigate through the system and perform actions to enter and review information it contains. The navigation component also presents messages to the user about the success or failure of his or her actions. The goal of the navigation system is to make the system as simple as possible to use. A good navigation component is one the user never really notices. It simply functions the way the user expects, and thus the user gives it little thought.

YOUR
TURN

12-6 Prototyping and Evaluation

Pretend you have been charged with the task of redesigning the interface for the ATM at your local bank. What

type of prototyping and interface evaluation approach would you recommend? Why?

⁷ A good source for usability testing is Jakob Nielsen, and Robert Mack (eds.) *Usability Inspection Methods* (New York: Wiley, 1994). See also www.useit.com/papers.

Basic Principles

One of the hardest things about using a computer system is learning how to manipulate the navigation controls to make the system do what you want. Analysts usually need to assume that users have not read the manual, have not attended training, and do not have external help readily at hand. All controls should be clear and understandable, and placed in an intuitive location on the screen. Ideally, the controls should anticipate what the user will do and simplify his or her efforts. For example, many setup programs are designed so that for a “typical” installation the user can simply keep pressing the “next” button.

Prevent Mistakes The first principle of designing navigation controls is to prevent the user from making mistakes. A mistake costs time and creates frustration. Worse still, a series of mistakes can cause the user to discard the system. Mistakes can be reduced by labeling commands and actions appropriately and by limiting choices. Too many choices can confuse the user, particularly when they are similar and hard to describe in the short space available on the screen. When there are many similar choices on a menu, consider creating a second level of menu or a series of options for basic commands.

Never display a command that cannot be used. For example, many Windows applications gray-out commands that cannot be used; they are displayed on pull-down menus in a very light-colored font, but they cannot be selected. This shows that they are available, but that they cannot be used in the current context. It also keeps all menu items in the same place.

When the user is about to perform a critical function that is difficult or impossible to undo (e.g., deleting a file), it is important to confirm the action with the user (and make sure the selection was not made by mistake). This is usually done by having the user respond to a confirmation message, which explains what the user has requested and asks the user to confirm that this action is correct.

Simplify Recovery from Mistakes No matter what the system designer does, users will make mistakes. The system should make it as easy as possible to correct these errors. Ideally, the system will have an Undo button that makes mistakes easy to override; however, writing the software for such buttons can be very complicated.

Use Consistent Grammar Order One of the most fundamental decisions is the *grammar order*. Most commands require the user to specify an object (e.g., file, record, word), and the action to be performed on that object (e.g., copy, delete). The interface can require the user to first choose the object and then the action (an *object-action order*), or first choose the action and then the object (an *action-object order*). Most Windows applications use an object-action grammar order (e.g., think about copying a block of text in your word processor).

The grammar order should be consistent throughout the system, both at the data element level as well as at the overall menu level. Experts debate about the advantages of one approach over the other, but because most users are familiar with the object-action order, most systems today are designed using that approach.

Types of Navigation Controls

There are two traditional hardware devices that can be used to control the user interface: the keyboard and a pointing device such as a mouse, trackball, or touch screen. In recent years, voice recognition systems have made an appearance, but they are not yet common. There are three basic software approaches for defining user commands: languages, menus, and direct manipulation.

Languages With a *command language*, the user enters commands using a special language developed for the computer system (e.g., UNIX and SQL both use command languages). Command languages sometimes provide greater flexibility than other approaches because the user can combine language elements in ways not predetermined by developers. However, they put a greater burden on users because users must learn syntax and type commands, rather than select from a well-defined, limited number of choices. Systems today use command languages sparingly, except in cases where there are an extremely large number of command combinations that make it impractical to try to build all combinations into a menu (e.g., SQL queries for databases).

Natural language interfaces are designed to understand the user's own language (e.g., English, French, Spanish). These interfaces attempt to interpret what the user means, and often they present back to the user a list of interpretations from which to choose. An example of the use of natural language is Microsoft's Office Assistant, which enables users to ask free-form questions for help.

Menus The most common type of navigation system today is the *menu*. A menu presents the user with a list of choices, each of which can be selected. Menus are easier to learn than languages because a limited number of available commands are presented to the user in an organized fashion. Clicking on an item with a pointing device or pressing a key that matches the menu choice (e.g., a function key) takes very little effort. Therefore, menus are usually preferred to languages.

Menu design needs to be done with care because the submenus behind a main menu are hidden from users until they click on the menu item. It is better to make menus broad and shallow (i.e., each menu containing many items with only one or two layers of menus) rather than narrow and deep (i.e., each menu containing only a few items, but each leading to three or more layers of menus). A broad and shallow menu presents the user with the most information initially so that he or she can see many options and only requires only a few mouse clicks or keystrokes to perform an action. A narrow and deep menu makes users hunt and seek for items hidden behind menu items and requires many more clicks or keystrokes to perform an action.

Research suggests that in an ideal world, any one menu should contain no more than eight items and it should take no more than two mouse clicks or keystrokes from any menu to perform an action (or three from the main menu that starts a system).⁸ However, analysts sometimes must break this guideline in the design of complex systems. In this case, menu items are often grouped together and separated by a horizontal line (see Figure 12-9). Often menu items have *hot keys* that enable experienced users to quickly invoke a command with keystrokes in lieu of a menu choice (e.g., Ctrl-F in Word invokes the Find command or Alt-F opens the file menu).

Menus should put together like items so that the user can intuitively guess what each menu contains. Most designers recommend grouping menu items by interface objects (e.g., customers, purchase orders, inventory) rather than by interface actions (e.g., new, update, format), so that all actions pertaining to one object are in one menu, all actions for another object are in a different menu, and so. However, this is highly dependent on the specific interface. As Figure 12-9 shows, Microsoft Word groups menu items by interface objects (e.g., File, Table, Window) and by interface actions (e.g., Edit, Insert, Format) on the same menu. Some of the more common types of menus include *menu bars*, *drop-down menus*, *pop-up menus*, *tab menus*, *tool bars*, and *image maps* (see Figure 12-10).

⁸ Kent L. Norman, *The Psychology of Menu Selection* (Norwood NJ.: Ablex Publishing Corp., 1991).

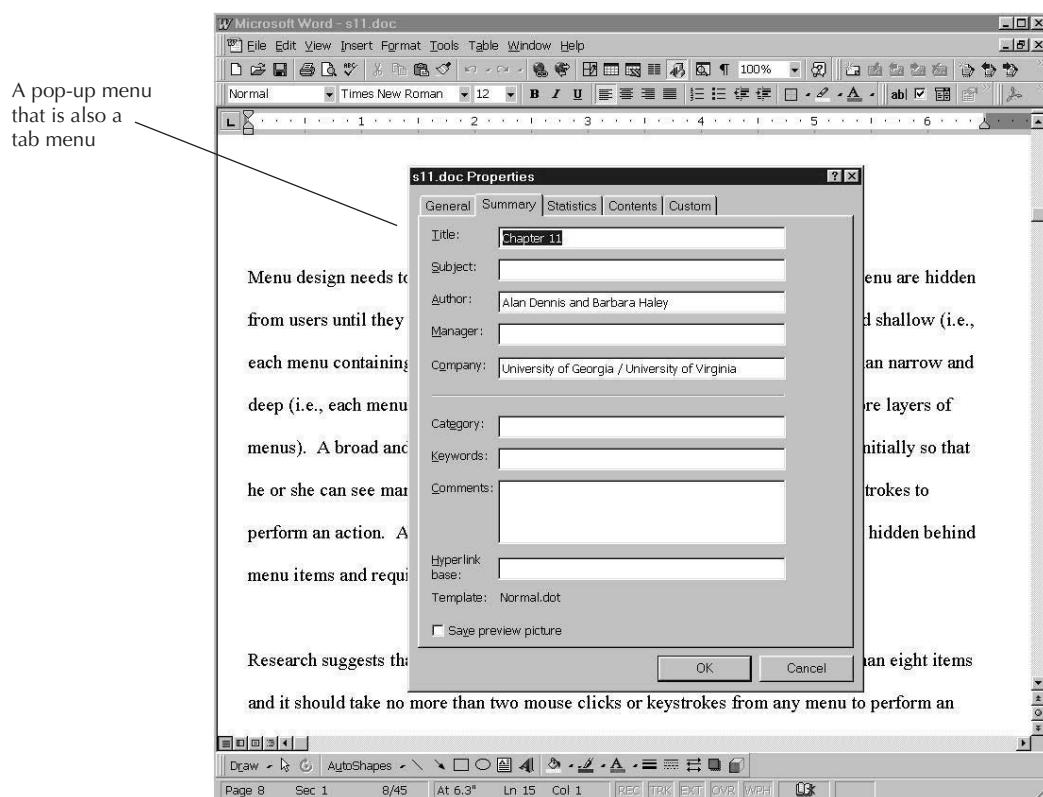
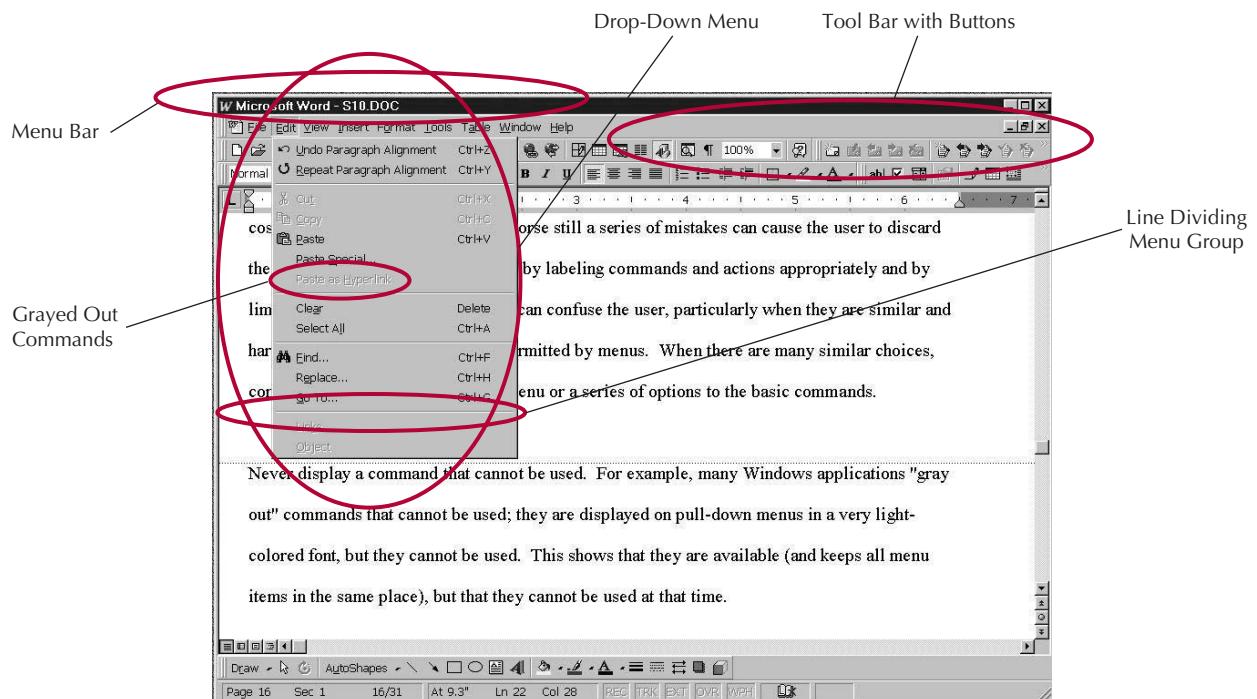


FIGURE 12-9 Common Types of Menus

Direct Manipulation With *direct manipulation*, the user enters commands by working directly with interface objects. For example, users can change the size of objects in Microsoft PowerPoint by clicking on them and moving their sides, or they can move files in Windows Explorer by dragging the filenames from one folder to another. Direct manipulation can be simple, but it suffers from two problems. First, users familiar with language- or menu-based interfaces don't always expect it. Second, not all commands are intuitive (e.g., how do you copy [not move] files in Windows explorer? On the Macintosh, why does moving a folder to the trash delete the file if it is on the hard disk, but eject the diskette if the file is on a diskette?).

Type of Menu	When to Use	Notes
Menu bar List of commands at the top of the screen; always on-screen	Main menu for system	Use the same organization as the operating system and other packages (e.g., File, Edit, View). Menu items are always one word, never two. Menu items lead to other menus rather than perform action. Never allow users to select actions they can't perform (instead, use grayed-out items).
Drop-down menu Menu that drops down immediately below another menu; disappears after one use	Second-level menu, often from menu bar	Menu items are often multiple words. Avoid abbreviations. Menu items perform action or lead to another cascading drop-down menu, pop-up menu, or tab menu.
Pop-up menu Menu that pops up and floats over the screen; disappears after one use	As a shortcut to commands for experienced users	Pop-up menus often (not always) invoked by a right click in Windows-based systems. These menus are often overlooked by novice users, so usually they should duplicate functionality provided in other menus.
Tab menu Multipage menu with one tab for each page that pops up and floats over the screen; remains on-screen until closed	When user needs to change several settings or perform several related commands	Menu items should be short to fit on the tab label. Avoid more than one row of tabs, because clicking on a tab to open it can change the order of the tabs and in virtually no other case does selecting from a menu rearrange the menu itself.
Tool bar Menu of buttons (often with icons) that remains on-screen until closed	As a shortcut to commands for experienced users	All buttons on the same tool bar should be the same size. If the labels vary dramatically in size, then use two different sizes (small and large). Buttons with icons should have a tool tip, an area that displays a text phrase explaining the button when the user pauses the mouse over it.
Image map Graphic image in which certain areas are linked to actions or other menus	Only when the graphic image adds meaning to the menu	The image should convey meaning to show which parts perform action when clicked. Tool tips can be helpful.

FIGURE 12-10 Types of Menus

YOUR TURN

12-7 Design a Navigation System

Design a navigation system for a system into which users must enter information about customers, products, and orders. For all three, users will want to change, delete, find one specific record, and list all records.

Messages

Messages are the way in which the system responds to a user and informs him or her of the status of the interaction. There are many different types of messages, such as *error messages*, *confirmation messages*, *acknowledgment messages*, *delay messages*, and *help messages* (see Figure 12-11). In general, messages should be clear, concise, and complete, which are sometimes conflicting objectives. All messages should be grammatically correct and free of jargon and abbreviations (unless they are the users' jargon and abbreviations). Avoid negatives because they can be confusing (e.g., replace "Are you sure you do not want to continue?" with "Do you want to quit?"). Likewise, avoid humor, because it wears off quickly after the same message appears dozens of times.

Messages should require the user to acknowledge them (by clicking for example), rather than being displayed for a few seconds and then disappearing. The exceptions are messages that inform the user of delays in processing, which should disappear once the delay has passed. In general, messages are text, but sometimes standard icons are used. For example, Windows displays an hourglass when the system is busy.

All messages should be carefully crafted, but error messages and help messages require particular care. Messages (and especially error messages) should always explain the problem in polite, succinct terms (e.g., what the user did incorrectly) and explain corrective action as clearly and as explicitly as possible so the user knows exactly what needs to be done. In the

Type of Messages	When to Use	Notes
Error message Informs the user that he or she has attempted to do something to which the system cannot respond	When user does something that is not permitted or not possible	Always explain the reason and suggest corrective action. Traditionally, error messages have been accompanied by a beep, but many applications now omit it or permit users to remove it.
Confirmation message Asks users to confirm that they really want to perform the action they have selected	When user selects a potentially dangerous choice, such as deleting a file	Always explain the cause and suggest possible action. Often include several choices other than "OK" and "cancel."
Acknowledgment message Informs the user that the system has accomplished what it was asked to do	Seldom or never. Users quickly become annoyed with all the unnecessary mouse clicks	Acknowledgment messages are typically included because novice users often like to be reassured that an action has taken place. The best approach is to provide acknowledgment information without a separate message on which the user must click. For example, if the user is viewing items in a list and adds one, then the updated list on the screen showing the added item is sufficient acknowledgment.
Delay message Informs the user that the computer system is working properly	When an activity takes more than seven seconds	Should permit the user to cancel the operation in case he or she does not want to wait for its completion. Should provide some indication of how long the delay may last.
Help message Provides additional information about the system and its components	In all systems	Help information is organized by table of contents and/or keyword search. Context-sensitive help provides information that is dependent on what the user was doing when help was requested. Help messages and on-line documentation are discussed in Chapter 14.

FIGURE 12-11 Types of Messages

case of complicated errors, the error message should display what the user entered, suggest probable causes for the error, and propose possible user responses. When in doubt, provide either more information than the user needs or the ability to get additional information. Error messages should provide a message number. Message numbers are not intended for users, but their presence makes it simpler for help desks and customer support lines to identify problems and help users because many messages use similar wording.

Navigation Design Documentation

The design of the navigation for a system is done through the use of window navigation diagrams (WND) and real use cases. Real use cases are derived from the essential use cases (see Chapter 6), use scenarios, and WNDs. Recall that an essential use case is one that only describes the minimum essential issues necessary to understand the required functionality. A real use case describes a specific set of steps that a user performs to use a specific part of a system. As such, real use cases are implementation dependent (i.e., they are detailed descriptions of how to use the system once it is implemented).

To evolve an essential use case into a real use case, two changes must be made. First, the use case type must be changed from essential to real. Second, all events must be specified in terms of the actual user interface. Therefore, the Normal Flow of Events, SubFlows, and Alternate/Exceptional Flows must be modified. The Normal Flow of Events, SubFlows, and Alternate/Exceptional Flows for the real use case associated the storyboard user interface prototype given in Figure 12-8 is shown in Figure 12-12. For example, Step 2 of the Normal Flow of Events states that “The System provides the Sales Rep with the Main Menu for the System,” which allows the Sales Rep to interact with the Maintain Client List aspect of the system.

CONCEPTS

12-B ERP User Interfaces Drive Users Nuts

IN ACTION

It used to take workers at Hydro Agri's Canadian Fertilizer stores about twenty seconds to process an order. After installing SAP, it now takes ninety seconds. Entering an order requires users to navigate through six screens to find the data fields that were on one screen in the old system. The problem became so critical during the spring planting rush that the project team installing the SAP system was pressed into service to take telephone orders.

Many other customers have complained about similar problems in SAP, and the other leading ERP systems. Ontario-based Algoma Steel uses PeopleSoft, and now has to use a dozen screens to enter employee data that were contained in two screens in their old custom-built personnel system. A-dec, a dental equipment maker based in Oregon, discovered the hard way that its Baan inventory system was still counting products that had been shipped in its on-hand inventories; the system required users to confirm the order shipments before the inventories were recorded as shipped—but didn't automatically take them to the confirmation screen.

So why have companies implemented ERP systems? The driving force behind most implementations was not

to simplify the users' jobs, but instead to improve the quality of the data, simplify system maintenance, and/or beat the Y2K problem. Ease-of-use wasn't a consideration, and what makes ERP systems so hard to use is that in the attempt to make them one-size-fits-all, developers had to include many little-used data items and processes. Instead of having a small custom system collecting only the data needed by the company itself (which could be condensed to fit one or two screens), companies now find themselves using a system designed to collect all the data items that any company could possibly use—data items that now require six to twelve screens.

Source: “ERP user interfaces drive workers nuts,” *ComputerWorld* (November 2, 1998).

Question:

1. Suppose you were a systems analyst at one of the leading ERP vendors (e.g., SAP, PeopleSoft, Baan). How could you apply the interface design principles and techniques in this chapter to improve the ease of use of your system?

INPUT DESIGN

Inputs mechanisms facilitate the entry of data into the computer system, whether highly structured data, such as order information (e.g., item numbers, quantities, costs) or unstructured information (e.g., comments). Input design means designing the screens used to enter the information, as well as any forms on which users write or type information (e.g., timecards, expense claims).

Use-Case Name: <i>Maintain Client List</i>	ID: 12	Importance Level: High
Primary Actor: <i>Sales Rep</i>	Use-Case Type: <i>Detail, Real</i>	
Stakeholders and Interests: <i>Sales Rep - wants to add, find or list clients</i>		
Brief Description: <i>This use case describes how sales representatives can search and maintain the client list.</i>		
Trigger: <i>Patient calls and asks for a new appointment or asks to cancel or change an existing appointment.</i>		
Type: <i>External</i>		
Relationships:		
Association: <i>Sales Rep</i>		
Include:		
Extend:		
Generalization:		
Normal Flow of Events:		
<ol style="list-style-type: none"> 1. The Sales Rep starts up the system. 2. The System provides the Sales Rep with the Main Menu for the System. 3. The System asks Sales Rep if he or she would like to add a client, find an existing Client, or to list all existing clients. <ul style="list-style-type: none"> If the Sales Rep wants to add a client, they click on the Add Client Link and execute G-1: New Client. If the Sales Rep wants to find a client, they click on the Find Client Link and execute G-2: Find Client. If the Sales Rep wants to list all clients, they click on the List Client Link and execute G-3: List Clients. 4. The System returns the Sales Rep to the Main Menu of the System. 		
Subflows:		
G-1: New Client		
<ol style="list-style-type: none"> 1. The System asks the Sales Rep for relevant information. 2. The Sales Rep types in the relevant information into the Form. 3. The Sales Rep submits the information to the System. 		
G-2: Find Client		
<ol style="list-style-type: none"> 1. The System asks the Sales Rep for the search information. 2. The Sales Rep types in the search information into the Form. 3. The Sales Rep submits the information to the System. 4. If the System finds a single Client that meets the search information, the System produces a Client Information report and returns the Sales Rep to the Main Menu of the System. 		
Else If the System finds a list of Clients that meet the search information, the System executes G-3: List Clients.		
G-3: List Clients		
<ol style="list-style-type: none"> 1. If this Subflow is executed from Step 3 The System creates a List of All clients. 		
Else		
<ol style="list-style-type: none"> 1. The System creates a List of clients that matched the G-2: Find Client search criteria. 2. The Sales Rep selects a client. 3. The System produces a Client Information report. 		
Alternate/Exceptional Flows:		
G-2 4a. The System produces an Error Message.		

FIGURE 12-12
Example Real Use Case

Basic Principles

The goal of the input mechanism is to simply and easily capture accurate information for the system. The fundamental principles for input design reflect the nature of the inputs (whether batch or online) and ways to simplify their collection.

Online versus Batch Processing There are two general formats for entering inputs into a computer system: online processing and batch processing. With *online processing* (sometimes called *transaction processing*), each input item (e.g., a customer order, a purchase order) is entered into the system individually, usually at the same time as the event or transaction prompting the input. For example, when you take a book out from the library, buy an item at the store, or make an airline reservation, the computer system that supports that process uses online processing to immediately record the transaction in the appropriate database(s). Online processing is most commonly used when it is important to have *real-time information* about the business process. For example, when you reserve an airline seat, the seat is no longer available for someone else to use.

With *batch processing*, all the inputs collected over some time period are gathered together and entered into the system at one time in a batch. Some business processes naturally generate information in batches. For example, most hourly payrolls are done using batch processing because time cards are gathered together in batches and processed at once. Batch processing also is used for transaction processing systems that do not require real-time information. For example, most stores send sales information to district offices so that new replacement inventory can be ordered. This information could be sent in real-time as it is captured in the store so that the district offices are aware within a second or two that a product is sold. If stores do not need this up-to-the-second real-time data, they will collect sales data throughout the day and transmit it every evening in a batch to the district office. This batching simplifies the data communications process and often saves in communications costs, but does mean that inventories are not accurate in real time, but rather are accurate only at the end of the day after the batch has been processed.

Capture Data at the Source Perhaps the most important principle of input design is to capture the data in an electronic format at its original source or as close to the original source as possible. In the early days of computing, computer systems replaced traditional manual systems that operated on paper forms. As these business processes were automated, many of the original paper forms remained, either because no one thought to replace them or because it was too expensive to do so. Instead, the business process continued to contain manual forms that were taken to the computer center in batches to be typed into the computer system by a *data entry operator*.

Many business processes still operate this way today. For example, most organizations have expense claim forms that are completed by hand and submitted to an accounting department, which approves them and enters them into the system in batches. There are three problems with this approach. First, it is expensive because it duplicates work (the form is filled out twice, once by hand, once by keyboard⁹). Second, it increases processing time because the paper forms must be physically moved through the process. Third, it increases the cost and probability of error, because it separates the entry from the processing of information; someone may misread the handwriting on the input form, data could be entered incorrectly, or the original input may contain an error that invalidates the information.

⁹ Or in the case of the University of Georgia, three times: first by hand on an expense form, a second time when it is typed onto a new form for the “official” submission because the accounting department refuses hand-written forms, and finally when it is typed into the accounting computer system.

Most transaction processing systems today are designed to capture data at its source. *Source data automation* refers to using special hardware devices to automatically capture data without requiring anyone to type it. Stores commonly use *bar code readers* that automatically scan products and that enter data directly into the computer system. No intermediate formats such as paper forms are used. Similar technologies include *optical character recognition*, which can read printed numbers and text (e.g., on checks), *magnetic stripe readers*, which can read information encoded on a stripe of magnetic material similar to a diskette (e.g., credit cards), and *smart cards* that contain micro-processors, memory chips, and batteries (much like credit-card sized calculators). As well as reducing the time and cost of data entry, these systems reduce errors because they are far less likely to capture data incorrectly. Today, portable computers and scanners allow data to be captured at the source even when in mobile settings (e.g., air courier deliveries, use of rental cars).

These automatic systems are not capable of collecting a lot of information, so the next best option is to capture data immediately from the source using a trained entry operator. Many airline and hotel reservations, loan applications, and catalog orders are recorded directly into a computer system while the customer provides the operator with answers to questions. Some systems eliminate the operator altogether and allow users to enter their own data. For example, several universities (e.g., the Massachusetts Institute of Technology [MIT]) no longer accept paper-based applications for admissions; all applications are typed by students into electronic forms.

The forms for capturing information (on a screen, on paper, etc.) should support the data source. That is, the order of the information on the form should match the natural flow of information from the data source, and data entry forms should match paper forms used to initially capture the data.

Minimize Keystrokes Another important principle is to minimize keystrokes. Keystrokes cost time and money, whether they are performed by a customer, user, or trained data-entry operator. The system should never ask for information that can be obtained in another way (e.g., by retrieving it from a database or by performing a calculation). Likewise, a system should not require a user to type information that can be selected from a list; selecting reduces errors and speeds entry.

In many cases, some fields have values that often recur. These frequent values should be used as the *default value* for the field so that the user can simply accept the value and not have to retype it time and time again. Examples of default values are the current date, the area code held by the majority of a company's customers, and a billing address that is based on the customer's residence. Most systems permit changes to default values to handle data entry exceptions as they occur.

YOUR TURN

12-8 Career Services

Pretend that you are designing the new interface for a Career Services system at your university that accepts student resumes and presents them in a standard format to recruiters. Describe how you could incorporate the basic

principles of input design into your interface design. Remember to include the use of online versus batch data input, the capture of information, and plans to minimize keystrokes.

Types of Inputs

Each data item that has to be input is linked to a field on the form into which its value is typed. Each field also has a field label, which is the text beside, above, or below the field that tells the user what type of information belongs in the field. Often the field label is similar to the name of the data element, but they do not have to have identical words. In some cases, a field will display a template over the entry box to show the user exactly how data should be typed. There are many different types of inputs, in the same way that there are many different types of fields (see Figure 12-13).

Text As the name suggests, a *text box* is used to enter text. Text boxes can be defined to have a fixed length or can be scrollable and can accept a virtually unlimited amount of text. In either case, boxes can contain single or multiple lines of textual information. Never use a text box if you can use a selection box.

Text boxes should have field labels placed to the *left* of the entry area their size clearly delimited by a box (or a set of underlines in a non-GUI interface). If there are multiple text boxes, their field labels and the left edges of their entry boxes should be aligned. Text boxes should permit standard GUI functions such as cut, copy, and paste.

Numbers A *number box* is used to enter numbers. Some software can automatically format numbers as they are entered, so that 3452478 becomes \$34,524.78. Dates are a special form of numbers that sometimes have their own type of number box. Never use a number box if you can use a selection box.

Selection Box A *selection box* enables the user to select a value from a predefined list. The items in the list should be arranged in some meaningful order, such as alphabetical for long lists, or in order of most frequently used. The default selection value should be chosen with care. A selection box can be initialized as “unselected.” However, it is better to start with the most commonly used item already selected.

There are six commonly used types of selection boxes: *check boxes*, *radio buttons*, *on-screen list boxes*, *drop-down list boxes*, *combo boxes*, and *sliders* (see Figure 12-14). The choice among the types of text selection boxes generally comes down to one of screen space and the number of choices the user can select. If screen space is limited and only one item can be selected, then a drop-down list box is the best choice, because not all list items need to be displayed on the screen. If screen space is limited but the user can select multiple items, an on-screen list box that displays only a few items can be used. Check boxes (for multiple selections) and radio buttons (for single selections) both require all list items to be displayed at all times, thus requiring more screen space, but since they display all choices, they are often simpler for novice users.

Input Validation

All data entered into the system need to be validated to ensure their accuracy. Input *validation* (also called *edit checks*) can take many forms. Ideally, computer systems should not accept data that fail any important validation check to prevent invalid information from entering the system. However, this can be very difficult, and invalid data often slip by data entry operators and the users providing the information. It is up to the system to identify invalid data and either make changes or notify someone who can resolve the information problem.

There are six different types of validation checks: *completeness check*, *format check*, *range check*, *check digit check*, *consistency check*, and *database check* (see Figure 12-15). Every system should use at least one validation check on all entered data, and ideally will perform all appropriate checks where possible.

OUTPUT DESIGN

Outputs are the reports that the system produces, whether on the screen, on paper, or in other media, such as the Web. Outputs are perhaps the most visible part of any system because a primary reason for using an information system is to access the information that it produces.

The screenshot shows a "Sample Input Form" window in Netscape Communicator. The form includes:

- Name:** A text input field.
- What is your major:** (Check one only) Radio buttons for MIS, Accounting, Marketing, Computer Science, and Management. MIS is selected.
- What software do you feel comfortable using:** (Check all that apply) Check boxes for Word, WordPerfect, Excel, Lotus 1-2-3, and Access. Word and WordPerfect are checked.
- Select where you were born:** An on-screen list box showing regions of Canada and the U.S., with "Eastern Canada" selected.
- Hair Color:** A drop-down list box with options: Brown, Blonde, Black, Red.
- Interest Score:** A slider with a scale from 0 to 100, currently set at 50.

FIGURE 12-13 Types of Input Boxes

Type of Box	When to Use	Notes
Check box Presents a complete list of choices, each with a square box in front	When several items can be selected from a list of items	Check boxes are not mutually exclusive. Do not use negatives for box labels. Check box labels should be placed in some logical order, such as that defined by the business process, or failing that, alphabetically or most commonly used first. Use no more than ten check boxes for any particular set of options. If you need more boxes, group them into subcategories.
Radio button Presents a complete list of mutually exclusive choices, each with a circle in front	When only one item can be selected from a set of mutually exclusive items	Use no more than six radio buttons in any one list; if you need more, use a drop-down list box. If there are only two options, one check box is usually preferred to two radio buttons, unless the options are not clear. Avoid placing radio buttons close to check boxes to prevent confusion between different selection lists.
On-screen list box Presents a list of choices in a box	Seldom or never—only if there is insufficient room for check boxes or radio buttons	This type of box can permit only one item to be selected (in which case it is an ugly version of radio buttons). This type of box can also permit many items to be selected (in which case it is an ugly version of check boxes), but users often fail to realize they can choose multiple items. This type of box permits the list of items to be scrolled, thus reducing the amount of screen space needed.
Drop-down list box Displays selected item in one-line box that opens to reveal list of choices	When there is insufficient room to display all choices	This type of box acts like radio buttons but is more compact. This type of box hides choices from users until it is opened, which can decrease ease of use; conversely, because it shelters novice users from seldom-used choices, it can improve ease of use. This type of box simplifies design if the number of choices is unclear, because it takes only one line when closed.
Combo box A special type of drop-down list box that permits user to type as well as scroll the list	Shortcut for experienced users	This type of box acts like drop-down list but is faster for experienced users when the list of items is long.
Slider Graphic scale with a sliding pointer to select a number	Entering an approximate numeric value from a large continuous scale	The slider makes it difficult for the user to select a precise number. Some sliders also include a number box to enable the user to enter a specific number.

FIGURE 12-14 Types of Selection Boxes

YOUR TURN

12-9 Career Services

Consider a Web form that a student would use to input student information and resume information into a Career Services application at your university. First, sketch out how this form would look and identify the fields that the

form would include. What types of validity checks would you use to make sure that the correct information is entered into the system?

Type of Validation	When to Use	Notes
Completeness check Ensures all required data have been entered	When several fields must be entered before the form can be processed	If required information is missing, the form is returned to the user unprocessed.
Format check Ensures data are of the right type (e.g., numeric) and in the right format (e.g., month, day, year)	When fields are numeric or contain coded data	Ideally, numeric fields should not permit users to type text data, but if this is not possible, the entered data must be checked to ensure it is numeric. Some fields use special codes or formats (e.g., license plates with three letters and three numbers) that must be checked.
Range check Ensures numeric data are within correct minimum and maximum values	With all numeric data, if possible	A range check permits only numbers between correct values. Such a system can also be used to screen data for “reasonableness”—e.g., rejecting birthdates prior to 1880 because people do not live to be a great deal over 100 years old (most likely, 1980 was intended).
Check digit check Check digits are added to numeric codes	When numeric codes are used	Check digits are numbers added to a code as a way of enabling the system to quickly validate correctness. For example, U.S. Social Security numbers and Canadian Social Insurance numbers assign only eight of the nine digits in the number. The ninth number—the check digit—is calculated using a mathematical formula from the first eight numbers. When the identification number is typed into a computer system, the system uses the formula and compares the result with the check digit. If the numbers don't match, then an error has occurred.
Consistency checks Ensure combinations of data are valid	When data are related	Data fields are often related. For example, someone's birth year should precede the year in which he or she was married. Although it is impossible for the system to know which data are incorrect, it can report the error to the user for correction.
Database checks Compare data against a database (or file) to ensure they are correct	When data are available to be checked	Data are compared against information in a database (or file) to ensure they are correct. For example, before an identification number is accepted, the database is queried to ensure that the number is valid. Because database checks are more “expensive” than the other types of checks (they require the system to do more work), most systems perform the other checks first and perform database checks only after the data have passed the previous checks.

FIGURE 12-15 Types of Input Validation

Basic Principles

The goal of the output mechanism is to present information to users so they can accurately understand it with the least effort. The fundamental principles for output design reflect how the outputs are used and ways to make it simpler for users to understand them.

Understand Report Usage The first principle in designing reports is to understand how they are used. Reports can be used for many different purposes. In some cases—but not very often—reports are read cover to cover because all information is needed. In most cases, reports are used to identify specific items or used as references to find information, so the order in which items are sorted on the report or grouped within categories is critical. This is particularly important for

the design of electronic or Web-based reports. Web reports that are intended to be read end-to-end should be presented in one long scrollable page, while reports that are primarily used to find specific information should be broken into multiple pages, each with a separate link. Page numbers and the date on which the report was prepared also are important for reference reports.

The frequency of the report may also play an important role in its design and distribution. *Real-time reports* provide data that are accurate to the second or minute at which they were produced (e.g., stock market quotes). *Batch reports* are those that report historical information that may be months, days, or hours old, and they often provide additional information beyond the reported information (e.g., totals, summaries, historical averages).

There are no inherent advantages to real-time reports over batch reports. The only advantages lie in the time value of the information. If the information in a report is time critical (e.g., stock prices, air traffic control information) then real time reports have value. This is particularly important because real-time reports often are expensive to produce; unless they offer some clear business value, they may not be worth the extra cost.

Manage Information Load Most managers get too much information, not too little (i.e., the *information load* that the manager must deal with is too great). The goal of a well-designed report is to provide all of the information needed to support the task for which it was designed. This does not mean that the report needs to provide all the information available on the subject—just what the users decide they need in order to perform their jobs. In some cases, this may result in the production of several different reports on the same topics for the same users because they are used in different ways. This is not a bad design.

For users in westernized countries, the most important information should always be presented first in the top-left corner of the screen or paper report. Information should be provided in a format that is usable without modification. The user should not need to re-sort the report's information, highlight critical information to find it more easily amid a mass of data, or perform additional mathematical calculations.

Minimize Bias No analyst sets out to design a biased report. The problem with bias is that it can be very subtle; analysts can introduce it unintentionally. *Bias* can be introduced by the way in which lists of data are sorted because entries that appear first in a list may receive more attention than those later in the list. Data often are sorted in alphabetical order, making those entries starting with the letter A more prominent. Data can be sorted in chronological order (or reverse chronological order), placing more emphasis on older (or most recent) entries. Data may be sorted by numeric value, placing more emphasis on higher or lower values. For example, consider a monthly sales report by state. Should the report be listed in alphabetical order by state name, in descending order by the amount sold, or in some other order (e.g., geographic region)? There are no easy answers to this, except to say that the order of presentation should match the way in which the information is used.

Graphical displays and reports can present particularly challenging design issues.¹⁰ The scale on the axes in graphs is particularly subject to bias. For most types of graphs, the scale should always begin at zero; otherwise, comparisons among values can be misleading. For example, in Figure 12-16, have sales increased by very much since 1993? The numbers in both charts are the same, but the visual images the two present are quite different. A glance at Figure 12-16(a) would suggest only minor changes, while a glance at Figure 12-16(b) might suggest that there have been some significant increases. In fact, sales have increased by a total of 15 percent over five years, or 3 percent per year. Figure 12-16(a) presents the most accurate picture; Figure 12-16(b) is biased because the scale starts very close to the lowest

¹⁰ Some of the best books on the design of charts and graphical displays are by Edward R. Tufte, *The Visual Display of Quantitative Information* (Cheshire, CT: Graphics Press, 1983); Edward R. Tufte, *Envisioning Information* (Cheshire, CT: Graphics Press, 1990); Edward R. Tufte, *Visual Explanations* (Cheshire, CT: Graphics Press, 1997); and William Cleveland, *Visualizing Data* (Summit, NJ: Hobart Press, 1993).

value in the graph and misleads the eye into inferring that there have been major changes (i.e., more than doubling from “two lines” in 1993 to “five lines” in 1998). Figure 12-16(b) is the default graph produced by Microsoft Excel.

Types of Outputs

There are many different types of reports, such as *detail reports*, *summary reports*, *exception reports*, *turnaround documents* and *graphs* (see Figure 12-17). Classifying reports is challenging because many reports have characteristics of several different types. For example, some detail reports also produce summary totals, making them summary reports.

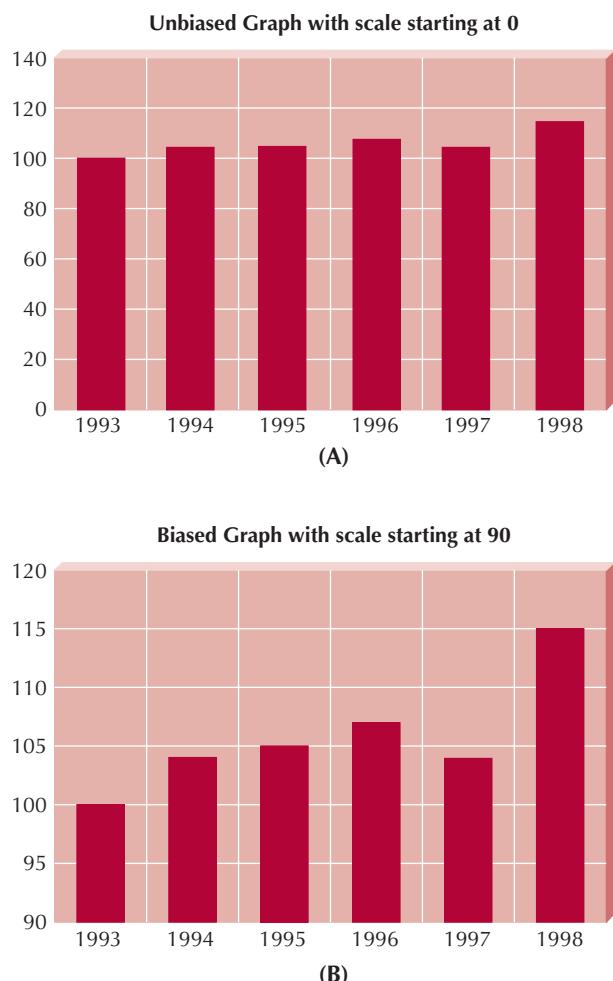


FIGURE 12-16
Bias in Graphs

**YOUR
TURN**

12-10 Finding Bias

Read through recent copies of a newspaper or popular press magazine, such as *Time*, *Newsweek*, or

BusinessWeek, and find four graphs. Which of these are biased? Why?

CONCEPTS

IN ACTION

12-C Selecting the Wrong Students

I helped a university department develop a small decision support system to analyze and rank students who applied to a specialized program. Some of the information was numeric and could easily be processed directly by the system (e.g., grade point average, standardized test scores). Other information required the faculty to make subjective judgments among the students (e.g., extracurricular activities, work experience). The users entered their evaluations of the subjective information via several data analysis screens in which the students were listed in alphabetical order.

To make the system easier to use, it was designed so that the reports listing the results of the analysis were also presented in alphabetical order by student name rather than in order from the highest ranked student to the lowest ranked student. In a series of tests prior to installation, the users selected the wrong students to

admit in 20 percent of the cases. They assumed, wrongly, that the students listed first were the highest ranked students and simply selected the first students on the list for admission. Neither the title on the report nor the fact that all the students' names were in alphabetical order made users realize that they had read the report incorrectly.

—Alan Dennis

Question:

1. This system was biased because users assumed that the list of students implied ranking. Pretend that you are an analyst charged with minimizing bias in this application. Where else may you find bias in the application? How would you eliminate it?

Type of Reports	When to Use	Notes
Detail report Lists detailed information about all the items requested	When user needs full information about the items	This report is usually produced only in response to a query about items matching some criteria. This report is usually read cover to cover to aid understanding of one or more items in depth.
Summary report Lists summary information about all items	When user needs brief information on many items	This report is usually produced only in response to a query about items matching some criteria, but it can be a complete database. This report is usually read for the purpose of comparing several items to each other. The order in which items are sorted is important.
Turnaround document Outputs that "turn around" and become inputs	When a user (often a customer) needs to return an output to be processed	Turnaround documents are a special type of report that are both outputs and inputs. For example, most bills sent to consumers (e.g., credit card bills) provide information about the total amount owed and also contain a form that consumers fill in and return with payment.
Graphs Charts used in addition to and instead of tables of numbers	When users need to compare data among several items	Well-done graphs help users compare two or more items or understand how one has changed over time. Graphs are poor at helping users recognize precise numeric values and should be replaced by or combined with tables when precision is important. Bar charts tend to be better than tables of numbers or other types of charts when it comes to comparing values between items (but avoid three-dimensional charts that make comparisons difficult). Line charts make it easier to compare values over time, whereas scatter charts make it easier to find clusters or unusual data. Pie charts show proportions or the relative shares of a whole.

FIGURE 12-17 Types of Reports

Media

There are many different types of media used to produce reports. The two dominant media in use today are paper and electronic. Paper is the more traditional media. For almost as long as there have been human organizations, there have been reports on paper or similar media (e.g., papyrus, stone). Paper is permanent, easy to use, and accessible in most situations. It also is highly portable, at least for short reports.

Paper also has several rather significant drawbacks. It is inflexible. Once the report is printed, it cannot be sorted or reformatted to present a different view of the information. Likewise, if the information on the report changes, the entire report must be reprinted. Paper reports are expensive, hard to duplicate, and require considerable supplies (paper, ink) and storage space. Paper reports also are hard to quickly move long distances (e.g., from a head office in Toronto to a regional office in Bermuda).

Many organizations are therefore moving to electronic production of reports, whereby reports are “printed,” but stored in electronic format on file servers or Web servers so that users can easily access them. Often the reports are available in more predesigned formats than their paper-based counterparts because the cost of producing and storing different formats is minimal. Electronic reports also can be produced on demand as needed, and they enable the user to more easily search for certain words. Furthermore, electronic reports can provide a means to support ad-hoc reports where users customize the contents of the report at the time the report is generated. Some users still print the electronic report on their own printers, but the reduced cost of electronic delivery over distance and the ease of enabling more users to access the reports than when they were only in paper form usually offsets the cost of local printing.

APPLYING THE CONCEPTS AT CD SELECTIONS

In the CD Selections example there are three different high-level use cases in the Internet sales system (see Figure 6-17): Maintain CD Information, Place Order, and Maintain Marketing Information. There are also six additional use cases, Maintain Order, Checkout, Create New Customer, Place InStore Hold, Place Special Order, and Fill Mail Order, associated with the Place Order use case. To keep the complexity of the current example under control, in this section, we focus only on the Place Order, Maintain Order, and Checkout use cases.

CONCEPTS

12-D Cutting Paper to Save Money

IN ACTION

One of the *Fortune* 500 firms with which I have worked had an eighteen-story office building for its world headquarters. It devoted two full floors of this building to nothing more than storing “current” paper reports (a separate warehouse was maintained outside the city for “archived” reports such as tax documents). Imagine the annual cost of office space in the headquarters building tied up in these paper reports. Now imagine how a staff member would gain access to the reports, and you can quickly understand the driving force behind electronic reports, even if most users end up printing them. Within one year

of switching to electronic reports (for as many reports as practical) the paper report storage area was reduced to one small storage room.

—Alan Dennis

Question:

1. What types of reports are most suited to electronic format? What types of reports are less suited to electronic reports?

Use Scenario Development

The first step in the interface design process was to develop the key use scenarios for the Internet sales system. Alec Adams, senior systems analyst at CD Selections and project manager for the Internet sales system, began by examining the essential use cases (see Figure 6-16) and thinking about the types of users and how they would interact with the system. To begin with, Alec identified two use scenarios: the browsing shopper and the hurry-up shopper (see Figure 12-6).¹¹ Alec also thought of several other use scenarios for the Web site in general, but omitted them since they were not relevant to the Internet sales portion. Likewise, he thought of several use scenarios that did not lead to sales (e.g., fans looking for information about their favorite artists and albums), and omitted them as well.

Interface Structure Design

Next, Alec created a window navigation diagram (WND) for the Web system. He began with the Place Order, Maintain Order, and Checkout essential use cases to ensure that all functionality defined for the system was included in the WND. Figure 12-18 shows the WND for the Web portion of the Internet Sales System. The system will start with a home page that contains the main menu for the sales system. Based on the essential use cases, Alec identified four basic operations that he felt made sense to support on the main menu: search the CD catalog, search by music category, review the contents of the shopping cart, and to actually place the order. Each of these was modeled as a hyperlink on the home page.

Alec decided to model the full search option as a pop-up search menu that allowed the customer to choose to search the CD catalog based on artist, title, or composer. He further decided that a textbox would be required to allow the customer to type in the name of the artist, title, or composer depending on the type of search requested. Finally, he chose to use a button to submit the request to the system. After the “submit” button is pressed, the system produces a report that was composed of hyperlinks to the individual information on each CD. A CD report containing the basic information is generated by clicking on the hyperlink associated with the CD. On the basic report, Alec added buttons for choosing to find out additional information on the CD and to add the CD to the shopping cart. If the “Detail” Button is pressed, a detailed report containing the marketing information on the CD is produced. Finally, Alec decided to include a button on this report to add the CD to the shopping cart.

The second basic operation supported on the home page was to allow the user to search the CD catalog by category of music. Like the previous operation, Alec chose to model the category search with a pop up search menu. In this case, once the customer chose the category, the system would produce the report with the hyperlinks to the individual information on each CD. From that point on, the navigation would be identical to the previous searches.

The third operation supported was to review the contents of the shopping cart. In this case, Alec decided to model the shopping cart as a report that contained three types of hyperlinks; one for removing an individual CD from the shopping cart, one for removing all CDs from the shopping cart, and one for placing the order. The removal hyperlinks would remove the individual CD (or all of the CDs) from the shopping cart if the user would confirm the operation. The place order link would send the customer to an order form. Once the customer filled out the order form, the customer would press the order button. The system would then respond with an order confirmation message box.

¹¹ Of course, it may be necessary to modify the original essential use cases in light of these new subtypes of customer. Furthermore, the structural and behavioral models may have to be modified. Remember that object-oriented systems analysis and design follows a RAD-based approach to developing systems, as such, additional requirements can be uncovered at any time.

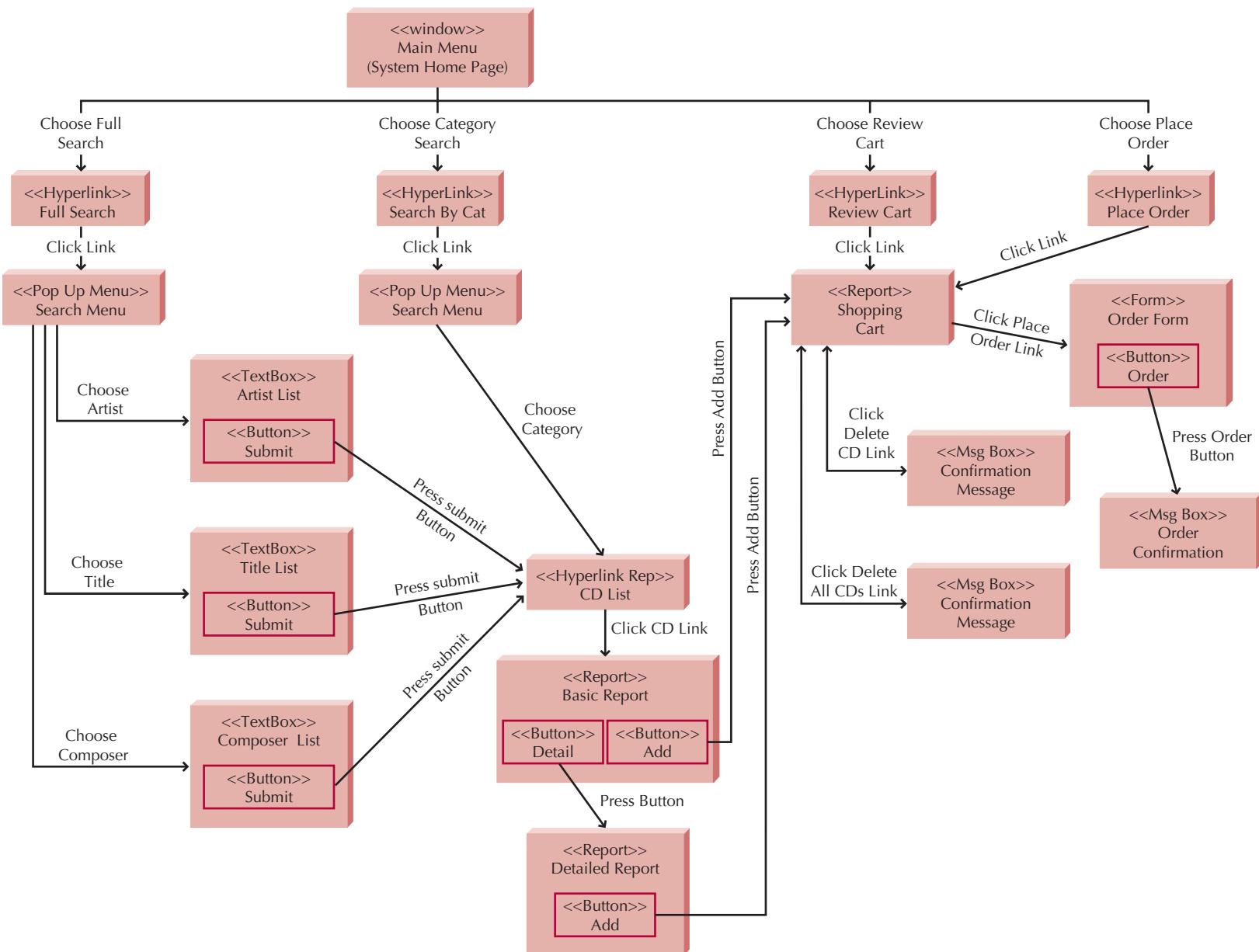


FIGURE 12-18 CD Selections Initial WND for the Web Portion of the Internet Sales System

The fourth operation supported on the home page was to allow the customer to place an order directly. Upon review, Alec decided that the place order and review cart operations were essentially identical. As such, he decided to force the user to have the Place Order and Review Shopping Cart operations go through the same process.

Alec also envisioned that by using frames, the user would be able to return to the home page from any screen. Documenting these would give the WND too many lines, so Alec simply put a note describing it with the WND.

The Revised WND Alec then examined the use scenarios to see how well the initial WND enabled different types of users to work through the system. He started with the Browsing Shopping use scenario and followed it through the WND, imagining what would appear on each screen and pretending to navigate through the system. He found the WND to work well, but he noticed a couple of minor issues related to the shopping cart. First, he decided that it would make sense to allow the customer to retrieve the information related to the CDs contained in the shopping cart. As such, he changed the stereotype of the user interface component from Report to HyperLink Rep and added a hyperlink from the Shopping Cart to the Basic Report created by the different search requests. Second, he noticed that the Shopping Cart was using hyperlinks to link to the Removal and Place Order processes. However, in all the other elements of the WND, he was using buttons to model the equivalent ideas. As such, he decided to change the Shopping Cart component to model these connections as buttons. Of course, this forced him to modify the transitions as well.

Alec next explored the Hurry-up Shopper use scenario. In this case, the WND did not work as well. Moving from the home page, to the search page, to the list of matching CDs, to the CD page with price and other information takes three mouse clicks. This falls within the three clicks rule, but for someone in a hurry, this may be too many. Alec decided to add a “quick-search” option to the home page that would enable the user to enter one search criteria (e.g., just artist name or title, rather than a more detailed search as would be possible on the search page) that would with one click take the user to the one CD that matched the criteria or to a list of CDs if there were more than one. This would enable an impatient user to get to the CD of interest in one or two clicks.

Once the CD is displayed on the screen, the Hurry-up Shopper use scenario would suggest that the user would immediately purchase the CD, do a new search, or abandon the Web site and surf elsewhere. This suggested two important changes. First, there had to be an easy way to go to the place order screen. As the WND stands (see Figure 12-18), the user must add the item to the shopping cart and then click on the link on the HTML frame to get to the place order screen. While the ability of users to notice the place order link in the frame would await the interface evaluation stage, Alec suspected, based on past experience, that a significant number of users would not see it. Therefore, he decided to add a button to the Basic Report screen and the Detailed Report screen called “Buy” (see Figure 12-19).

Second, since the Hurry-up Shopper might want to search for another CD instead of buying the CD, Alec decided to include the quick-search item from the home page on the frame. This would make all searches immediately available from anywhere in the system. This would mean that all functionality on the home page would now be carried on the frame. Alec updated the note on the bottom attached to the WND to reflect the change.

Finally, upon review of the WND, Alec decided to remodel the Artist List, Title List, and Composer Lists as window stereotypes instead of textbox stereotypes. He then added a Search Item textbox to each of these elements. Figure 12-19 shows the revised WND for the Web portion of the Place Order use case. All changes are highlighted.

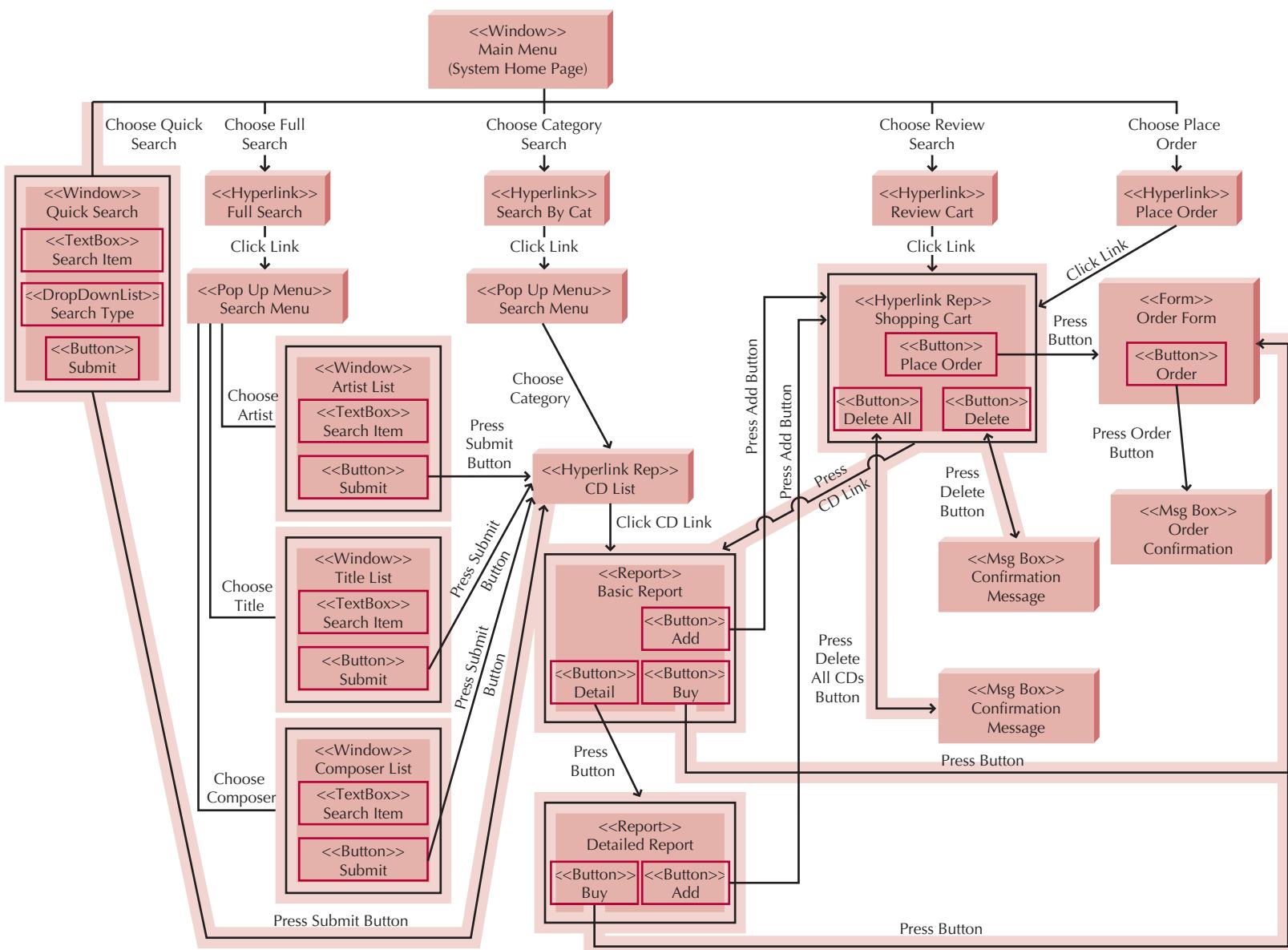


FIGURE 12-19 CD Selections Revised WND for the Web Portion of the Internet Sales System

Interface Standards Design

Once the WND was complete, Alec moved on to develop the interface standards for the system. The interface metaphor was straightforward: a CD Selections music store. The key interface objects and actions were equally straightforward, as was the use of the CD Selections logo icon. See Figure 12-20.

Interface Template Design

For the interface template, Alec decided on a simple, clean design that had a modern background pattern, with the CD-Selections logo in the upper-left corner. The template had two navigation areas: one menu across the top for navigation within the entire Web site (e.g., overall Web site home page, store locations) and one menu down the left edge for navigation within the Internet sales system. The left edge menu contained the links to the top-level operations (see WND in Figure 12-19), as well as the “quick search” option. The center area of the screen is used for displaying forms and reports when the appropriate operation is chosen (see Figure 12-21).

At this point, Alec decided to seek some quick feedback on the interface structure and standards before investing time in prototyping the interface designs. Therefore, he met with Margaret Mooney, the project sponsor, and Chris Campbell, the consultant, to discuss the emerging design. Making changes at this point would be much simpler than after doing the prototype. Margaret and Chris had a few suggestions, so after the meeting Alec made the changes and moved into the design prototyping step.

Design Prototyping

Alec decided to develop a hypertext mark-up language (HTML) prototype of the system. The Internet sales system was new territory for CD Selections and a strategic investment in a new business model, so it was important to make sure that no key issues were overlooked. The HTML prototype would provide the most detailed information and enable interactive evaluation of the interface.

Interface Metaphor: A CD Selections Music Store

Interface Objects

- **CD:** All items, whether CD, tape, or DVD, unless it is important to distinguish among them
- **Artist:** Person or group who records the CD
- **Title:** Title or name of CD
- **Composer:** Person or group who wrote the music for the CD (primarily used for classical music)
- **Music Category:** Type of music. Current categories include: Rock, Jazz, Classical, Country, Alternative, Soundtracks, Rap, Folk, Gospel.
- **CD List:** List of CD(s) that match the specified criteria
- **Shopping Cart:** Place to store CDs until they are requested

Interface Actions

- **Search for:** Display a CD list that matches specified criteria
- **Browse:** Display a CD list sorted in order by some criteria
- **Order:** Authorize special order or place hold

Interface Icons

- **CD Selections Logo:** Will be used on all screens

FIGURE 12-20
CD-Selection Interface Standards

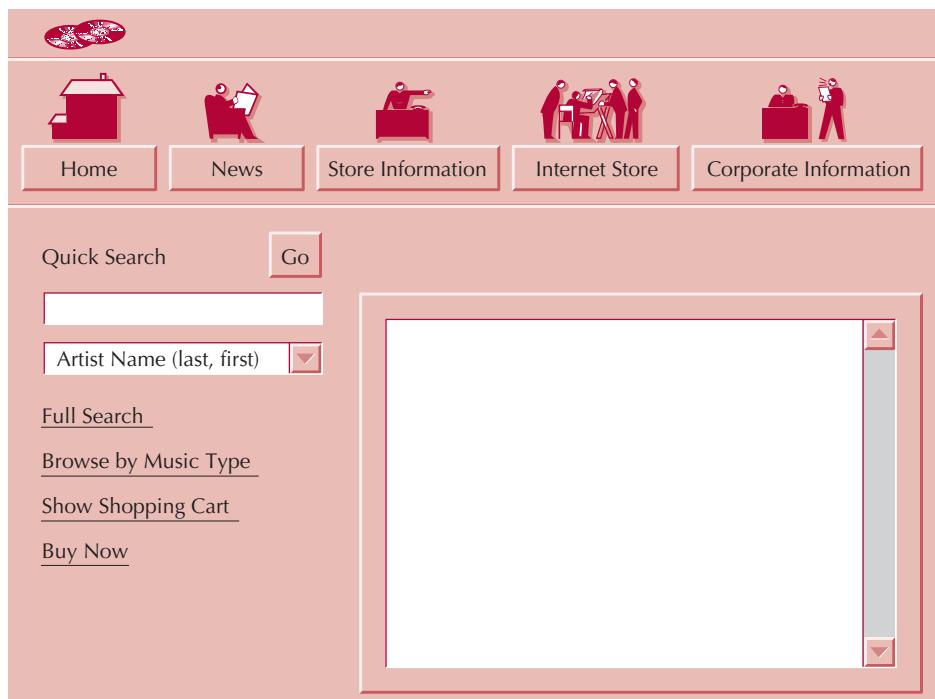


FIGURE 12-21
CD Selections Interface Template for the Web Portion of the Internet Sales System

In designing the prototype, Alec started with the home screen and gradually worked his way through all the screens. The process was very iterative and he made many changes to the screens as he worked. Once he had an initial prototype designed, he posted it on CD Selections intranet and solicited comments from several friends with lots of Web experience. He revised it based on the comments he received. Figure 12-22 presents some screens from the prototype.

Interface Evaluation

The next step was interface evaluation. Alec decided on a two-phase evaluation. The first evaluation was to be an interactive evaluation conducted by Margaret, her marketing managers, selected staff members, selected store managers, and Chris. They worked hands-on with the prototype and identified several ways to improve it. Alec modified the HTML prototype to reflect the changes suggested by the group and asked Margaret and Chris to review it again.

The second evaluation was another interactive evaluation, this time by a series of two focus groups of potential customers—one with little Internet experience, the other with extensive Internet experience. Once again, several minor changes were identified. Alec again modified the HTML prototype and asked Margaret and Chris to review it again. Once they were satisfied, the interface design was complete.

Navigation Design Documentation

The last step that Alec completed was to document the navigation design through the use of real use cases. To accomplish this, Alec gathered together the essential use case (see Figure 6-15), the use scenarios (see Figure 12-6), the window navigation diagram (see Figure 12-19), and the user interface prototype (see Figures 12-21 and 12-22). First, he copied the contents of the essential use case to the real use case. He changed the type from detail,



(A)



(B)

FIGURE 12-22
 Sample Interfaces from the CD Selections Design Prototype

essential to detail, real and the primary actor was specialized to browsing customer instead of simply customer. Second, he wrote the specific set of steps and responses that described the interaction between the browsing customer and system. Figure 12-23 shows a partial listing of the steps in the Normal Flow of Events and SubFlows sections of the real use case. Last, he repeated the steps for the hurry-up customer.

SUMMARY

User Interface Design Principles

The first element of the user interface design is the layout of the screen, form or report, which is usually depicted using rectangular shapes with a top area for navigation, a central area for inputs and outputs, and a status line at the bottom. The design should help the user

Use-Case Name: <i>Place Order</i>	ID: 15	Importance Level: <i>High</i>		
Primary Actor: <i>The Browsing Customer</i>	Use-Case Type:	<i>Detail, Real</i>		
Stakeholders and Interests:	<i>Customer Wants to search web site to purchase CD</i>	<i>EM Manager Wants to maximize customer satisfaction.</i>		
Brief Description: <i>This use case describes how customers can search the web site and place orders.</i>				
Trigger: <i>customer visits web site</i>				
Type:	<i>External</i>			
Relationships:				
Association: <i>Customer</i>				
Include: <i>checkout, Maintain Order</i>				
Extend:				
Generalization:				
Normal Flow of Events:				
1. The customer visits the Web site. 2. The System displays the Home Page if the customer wants to do a Full Search, execute G-1:Full Search if the customer wants to Browse by Music Type, execute G-2: Browse by Music Type if the customer wants to see any Special Deals, execute G-3: Special Deals if the customer wants to see the contents of the Shopping Cart, execute G-4: Shopping Cart if the customer wants to Buy Now, execute G-5: Buy Now 3. The Customer leaves the site.				
Subflows:				
G-1: Full Search 1. The Customer clicks the Full Search hyperlink 2. The System displays the search type pop-up menu if the customer chooses an Artist search, execute G-1a: Artist List if the customer chooses an Title search, execute G-1a: Title List if the customer chooses an Composer search, execute G-1a: composer List G-1a: Artist List 1. The System displays the Artist List window in the Center Area of the Home Page. 2. The Customer enters the Artist Name into the Search Item text box. 3. The Customer presses the Submit button. 4. The System executes G-2a: CD List. G-2a: CD List 1. The System displays the CD List hyperlink report. 2. The Customer chooses a CD to review by clicking the CD link. 3. The System executes G-2b: Display Basic Report 4. Iterate over steps 2 and 3.				
Alternate/Exceptional Flows:				

FIGURE 12-23
The Browsing Customer Real Use Case (Partial Listing Only)

**YOUR
TURN**

12-11 Completing the CD Selections Real Use Case

Based on the use case diagram (see Figure 6-17), the user interface prototype (see Figures 12-21 and 12-22), the use scenarios (see Figure 12-6), and the WND (see Figure 12-19) for the Web portion of CD Selections Internet sales

system, complete the real use cases. Be sure to add back the Place In Store Hold, Place Special Order, and Fill Mail Order use cases.

**YOUR
TURN**

12-12 Revising CD Selections WND

Based on the user interface prototype (see Figures 12-21 and 12-22) and the real use cases (see Your Turn 12-11) for

the Web portion of CD Selections Internet sales system, revise the window navigation diagram (see Figure 12-19).

be aware of content and context, both between different parts of the system as they navigate through it and within any one form or report. All interfaces should be aesthetically pleasing (not works of art) and need to include significant white space, use colors carefully, and be consistent with fonts. Most interfaces should be designed to support both novice/first time users, as well as experienced users. Consistency in design (both within the system and across other systems used by the users) is important for the navigation controls, terminology, and the layout of forms and reports. Finally, all interfaces should attempt to minimize user effort, for example, by requiring no more than three clicks from the main menu to perform an action.

The User Interface Design Process

First, analysts develop use scenarios that describe commonly used patterns of actions that the users will perform. Second, they design the interface structure via a WND based on the essential use cases. The WND is then tested with the use scenarios to ensure that it enables users to quickly and smoothly perform these scenarios. Third, analysts define the interface standards in terms of interface metaphor(s), objects, actions, and icons. These elements are drawn together by the design of a basic interface template for each major section of the system. Fourth, the designs of the individual interfaces are prototyped, either through a simple storyboard, an HTML prototype, or a prototype using the development language of the system itself (e.g., Visual Basic). Finally, interface evaluation is conducted using heuristic evaluation, walkthrough evaluation, interactive evaluation, or formal usability testing. This evaluation almost always identifies improvements, so the interfaces are redesigned and evaluated further.

Navigation Design

The fundamental goal of the navigation design is to make the system as simple to use as possible, by preventing the user from making mistakes, simplifying the recovery from mistakes, and using a consistent grammar order (usually object-action order). Command languages, natural languages, and direct manipulation are used in navigation, but the most

common approach is menus (menu bar, drop down menu, pop-up menu, tab menu, button and tool bars, and image maps). Error messages, confirmation messages, acknowledgment messages, delay messages, and help messages are common types of messages. Once the navigation design is agreed upon, it is documented in the form of windows navigation diagrams and real use cases.

Input Design

The goal of the input mechanism is to simply and easily capture accurate information for the system, typically by using online or batch processing, capturing data at the source, and minimizing keystrokes. Input design includes both the design of input screens and all preprinted forms that are used to collect data before it is entered into the information system. There are many types of inputs, such as text fields, number fields, check boxes, radio buttons, on-screen list boxes, drop-down list boxes, and sliders. Most inputs are validated by using some combination of completeness checks, format checks, range checks, check digit checks, consistency checks, and database checks.

Output Design

The goal of the output mechanism is to present information to users so they can accurately understand it with the least effort, usually by understanding how reports will be used and designing them to minimize information overload and bias. Output design means designing both screens and reports in other media, such as paper and the Web. There are many types of reports such as detail reports, summary reports, exception reports, turnaround documents, and graphs.

KEY TERMS

Acknowledgment message	Direct manipulation	Interactive evaluation
Action-object order	Drop-down list box	Interface action
Aesthetics	Drop-down menu	Interface design prototype
Bar code reader	Ease of learning	Interface evaluation
Batch processing	Ease of use	Interface icon
Batch report	Edit check	Interface metaphor
Bias	Error message	Interface object
Button	Essential use case	Interface standards
Check box	Exception report	Interface template
Check digit check	Field	Language prototype
Combo box	Field label	Layout
Command language	Form	Magnetic stripe readers
Completeness check	Format Check	Menu
Confirmation message	Grammar order	Menu bar
Consistency	Graph	Natural language
Consistency check	Graphical User Interface (GUI)	Navigation mechanism
Content awareness	Help message	Number box
Data entry operator	Heuristic evaluation	Object-action order
Database check	Hot key	Online processing
Default value	HTML prototype	On-screen list box
Delay message	Image map	Optical character recognition
Density	Information load	Output mechanism
Detail report	Input mechanism	Pop-up menu

Radio button	State	Usability testing
Range check	Stereotype	Use case
Real-time information	Storyboard	Use scenario
Real-time report	Summary report	User experience
Real use case	System interface	User interface
Report	Tab menu	Validation
Screen	Text box	Walkthrough evaluation
Selection box	Three clicks rule	White space
Sequence diagrams	Tool bar	Window
Slider	Transaction processing	Windows navigation diagram (WND)
Smart card	Transition	
Source data automation	Turnaround document	

QUESTIONS

1. Explain three important user interface design principles.
2. What are three fundamental parts of most user interfaces?
3. Why is content awareness important?
4. What is white space and why is it important?
5. Under what circumstances should densities be low? High?
6. How can a system be designed to be used by experienced and first-time users?
7. Why is consistency in design important? Why can too much consistency cause problems?
8. How can different parts of the interface be consistent?
9. Describe the basic process of user interface design.
10. What are use scenarios, and why are they important?
11. What is a windows navigation diagram (WND), and why is it used?
12. Why are interface standards important?
13. Explain the purpose and contents of interface metaphors, interface objects, interface actions, interface icons, and interface templates.
14. Why do we prototype the user interface design?
15. Compare and contrast the three types of interface design prototypes.
16. Why is it important to perform an interface evaluation before the system is built?
17. Compare and contrast the four types of interface evaluation.
18. Under what conditions is heuristic evaluation justified?
19. What type of interface evaluation did you perform in the Your Turn Box 12-1?
20. Describe three basic principles of navigation design.
21. How can you prevent mistakes?
22. Explain the differences between object-action order and action-object order.
23. Describe four types of navigation controls.
24. Why are menus the most commonly used navigation control?
25. Compare and contrast four types of menus.
26. Under what circumstances would you use a drop-down menu versus a tab menu?
27. Under what circumstances would you use an image map versus a simple list menu?
28. Describe five types of messages.
29. What are the key factors in designing an error message?
30. What is context-sensitive help? Does your word processor have context-sensitive help?
31. How do an essential use case and a real use case differ?
32. What is the relationship between essential use cases and use scenarios?
33. What is the relationship between real use cases and use scenarios?
34. Explain three principles in the design of inputs.
35. Compare and contrast batch processing and online processing. Describe one application that would use batch processing and one that would use online processing.
36. Why is capturing data at the source important?
37. Describe four devices that can be used for source data automation.
38. Describe five types of inputs.
39. Compare and contrast check boxes and radio buttons. When would you use one versus the other?
40. Compare and contrast on-screen list boxes and drop-down list boxes. When would you use one versus the other?
41. Why is input validation important?
42. Describe five types of input validation methods.
43. Explain three principles in the design of outputs.

44. Describe five types of outputs.
45. When would you use electronic reports rather than paper reports, and vice-versa?
46. What do you think are three common mistakes that novice analysts make in navigation design?
47. What do you think are three common mistakes that novice analysts make in input design?
48. What do you think are three common mistakes that novice analysts make in output design?
49. How would you improve the form in Figure 12-4?

EXERCISES

- A. Develop two use scenarios for Web site that sells some retail products (e.g., books, music, clothes).
- B. Draw a WND for a Web site that sells some retail products (e.g., books, music, clothes).
- C. Describe the primary components of the interface standards for a Web site that sells some retail products (metaphors, objects, actions, icons, and template).
- D. Based on the use case diagram in Exercise O in Chapter 6:
1. Develop two use scenarios.
 2. Develop the interface standards (omitting the interface template).
 3. Draw a WND.
 4. Design a storyboard.
- E. Based on your solution to Exercise D:
1. Design an interface template.
 2. Develop an HTML prototype.
 3. Develop a real use case.
- F. Based on the use case diagram in Exercise Q in Chapter 6:
1. Develop two use scenarios.
 2. Develop the interface standards (omitting the interface template).
 3. Draw a WND.
 4. Design a storyboard.
- G. Based on your solution to Exercise F:
1. Design an interface template.
 2. Develop an HTML prototype.
 3. Develop a real use case.
- H. Based on the use case diagram in Exercise S in Chapter 6:
- I. Develop two use scenarios.
- J. Based on your solution to Exercise H:
1. Design an interface template.
 2. Develop an HTML prototype.
 3. Develop a real use case.
- K. Based on the use case diagram in Exercise U in Chapter 6:
1. Develop two use scenarios.
 2. Develop the interface standards (omitting the interface template).
 3. Draw a WND.
 4. Design a storyboard.
- L. Based on your solution to Exercise J:
1. Design an interface template.
 2. Develop an HTML prototype.
 3. Develop a real use case.
- M. Ask Jeeves (<http://www.ask.com>) is an Internet search engine that uses natural language. Experiment with it and compare it to search engines that use key words.
- N. Draw a windows navigation diagram (WND) for the Your Turn Box 12-7 using the opposite grammar order from your original design (if you didn't do it, draw two WNDs, one in each grammar order). Which is best? Why?
- O. In the Your Turn Box 12-7, you probably used menus. Design it again using a command language.

MINICASES

1. Tots to Teens is a catalog retailer specializing in children's clothing. A project has been under way to develop a new order entry system for the company's catalog clerks. The old system had a character-based user interface that corresponded to the system's COBOL underpinnings. The new system will feature a graphical user interface more in keeping with up-to-date PC products in use today. The company hopes that this new user interface will help reduce the turnover they have experienced with their order entry

clerks. Many newly hired order entry staff found the old system very difficult to learn and were overwhelmed by the numerous mysterious codes that had to be used to communicate with the system.

A user interface walkthrough evaluation was scheduled for today to give the user a first look at the new system's interface. The project team was careful to invite several key users from the order entry department. In particular, Norma was included because of her years of experience with the order entry system.

Norma was known to be an informal leader in the department; her opinion influenced many of her associates. Norma had let it be known that she was less than thrilled with the ideas she had heard for the new system. Due to her experience and good memory, Norma worked very effectively with the character-based system and was able to breeze through even the most convoluted transactions with ease. Norma had trouble suppressing a sneer when she heard talk of such things as "icons" and "buttons" in the new user interface.

Cindy was also invited to the walkthrough because of her influence in the order entry department. Cindy has been with the department for just one year, but she quickly became known because of her successful organization of a sick child day-care service for the children of the department workers. Sick children are the number-one cause of absenteeism in the department, and many of the workers could not afford to miss workdays. Never one to keep quiet when a situation needed improvement, Cindy has been a vocal supporter of the new system.

- a. Drawing upon the design principles presented in the text, describe the features of the user interface that will be most important to experienced users like Norma.
 - b. Drawing upon the design principles presented in the text, describe the features of the user interface that will be most important to novice users like Cindy.
2. The members of a systems development project team have gone out for lunch together, and as often happens, the conversation turns to work. The team has been working on the development of the user interface design, and so far, work has been progressing smoothly. The team should be completing work on the interface prototypes early next week. A combination of storyboards and language prototypes has been used in this project. The storyboards depict the overall structure and flow of the system, but the team developed language prototypes of the actual screens because they felt that seeing the actual screens would be valuable for the users.

Chris (the youngest member of the project team): I read an article last night about a really cool way to evaluate a user interface design. It's called usability testing, and it's done by all the major software vendors. I think we should use it to evaluate our interface design.

Heather (system analyst): I've heard of that, too, but isn't it really expensive?

Mark (project manager): I'm afraid it is expensive, and I'm not sure we can justify the expense for this project.

Chris: But we really need to know that the interface

works. I thought this usability testing technique would help us prove we have a good design.

Amy (systems analyst): It would, Chris, but there are other ways too. I assumed we'd do a thorough walkthrough with our users and present the interface to them at a meeting. We can project each interface screen so that the users can see it and give us their reaction. This is probably the most efficient way to get the users' response to our work.

Heather: That's true, but I'd sure like to see the users sit down and work with the system. I've always learned a lot by watching what they do, seeing where they get confused, and hearing their comments and feedback.

Ryan (systems analyst): It seems to me that we've put so much work into this interface design that all we really need to do is review it ourselves. Let's just make a list of the design principles we're most concerned about and check it ourselves to make sure we've followed them consistently. If we have, we should be fine. We want to get moving on the implementation, you know.

Mark: These are all good ideas. It seems like we've all got a different view of how to evaluate the interface design. Let's try and sort out the technique that is best for our project.

Develop a set of guidelines that can help a project team like the one above select the most appropriate interface evaluation technique for their project.

3. The menu structure for Holiday Travel Vehicle's existing character-based system is shown below. Develop and prototype a new interface design for the system's functions using a graphical user interface. Also, develop a set of real use cases for your new interface. Assume the new system will need to include the same functions as those shown in the menus provided. Include any messages that will be produced as a user interacts with your interface (error, confirmation, status, etc.). Also, prepare a written summary that describes how your interface implements the principles of good interface design as presented in the textbook.

Holiday Travel Vehicles

Main Menu

- 1 Sales Invoice
- 2 Vehicle Inventory
- 3 Reports
- 4 Sales Staff

Type number of menu selection here:_____

Holiday Travel Vehicles

Sales Invoice Menu

- 1 Create Sales Invoice
- 2 Change Sales Invoice
- 3 Cancel Sales Invoice

Type number of menu selection here: _____

Holiday Travel Vehicles

Vehicle Inventory Menu

- 1 Create Vehicle Inventory Record
- 2 Change Vehicle Inventory Record
- 3 Delete Vehicle Inventory Record

Type number of menu selection here: _____

Holiday Travel Vehicles

Reports Menu

- 1 Commission Report
- 2 RV Sales by Make Report
- 3 Trailer Sales by Make Report
- 4 Dealer Options Report

Type number of menu selection here: _____

Holiday Travel Vehicles

Sales Staff Maintenance Menu

- 1 Add Salesperson Record
- 2 Change Salesperson Record
- 3 Delete Salesperson Record

Type number of menu selection here: _____

4. One aspect of the new system under development at Holiday Travel Vehicles will be the direct entry of the sales invoice into the computer system by the salesperson as the purchase transaction is being completed. In the current system, the salesperson fills out a paper form (shown on p. 422):

Design and prototype an input screen that will permit the salesperson to enter all the necessary information for the sales invoice. The following information may be helpful in your design process. Assume that Holiday Travel Vehicles sells recreational vehicles and trailers from four different manufacturers. Each manufacturer has a fixed number of names and models of RVs and trailers. For the purposes of your prototype, use this format:

Mfg-A	Name-1 Model-X
Mfg-A	Name-1 Model-Y
Mfg-A	Name-1 Model-Z
Mfg-B	Name-1 Model-X
Mfg-B	Name-1 Model-Y
Mfg-B	Name-2 Model-X
Mfg-B	Name-2 Model-Y
Mfg-B	Name-2 Model-Z
Mfg-C	Name-1 Model-X
Mfg-C	Name-1 Model-Y
Mfg-C	Name-1 Model-Z
Mfg-C	Name-2 Model-X
Mfg-C	Name-3 Model-X
Mfg-D	Name-1 Model-X
Mfg-D	Name-2 Model-X
Mfg-D	Name-2 Model-Y

Also, assume there are ten different dealer options that could be installed on a vehicle at the customer's request. The company currently has ten salespeople on staff.

<i>Holiday Travel Vehicles</i>			
Sales Invoice	Invoice #: _____ Invoice Date: _____		
Customer Name: _____			
Address: _____			
City: _____			
State: _____			
Zip: _____			
Phone: _____			
New RV/TRAILER			
(circle one)	Name: _____ Model: _____ Serial #: _____ Year: _____ Manufacturer: _____		
Trade-in RV/TRAILER			
(circle one)	Name: _____ Model: _____ Year: _____ Manufacturer: _____		
Options:	<u>Code</u>	<u>Description</u>	<u>Price</u>
	_____	_____	_____
	_____	_____	_____
	_____	_____	_____
	_____	_____	_____
Vehicle Base Cost:	_____		
Trade-in Allowance:	(Salesperson Name)		
Total Options:	_____		
Tax:	_____		
License Fee:	_____		
Final Cost:	(Customer Signature)		

CHAPTER 13

PHYSICAL ARCHITECTURE LAYER DESIGN

An important component of the design of an information system is the design of the physical architecture layer, which describes the system's hardware, software and network environment. The *physical architecture layer design* flows primarily from the nonfunctional requirements, such as operational, performance, security, cultural, and political requirements. The deliverable from the physical architecture layer design includes the infrastructure design and the hardware and software specification.

OBJECTIVES

- Understand the different physical architecture components.
- Understand server-based, client-based, and client server physical architectures.
- Be familiar with distributed objects computing.
- Be able to create a network model using a deployment diagram.
- Understand how operational, performance, security, cultural, and political requirements affect the design of the physical architecture layer.
- Be familiar with how to create a hardware and software specification.

CHAPTER OUTLINE

Introduction	The Network Model
Elements of the Physical Architecture	Nonfunctional Requirements and
Layer	Physical Architecture Layer Design
Architectural Components	Operational Requirements
Server-Based Architectures	Performance Requirements
Client-Based Architectures	Security Requirements
Client–Server Architectures	Cultural and Political Requirements
Client–Server Tiers	Synopsis
Distributed Objects Computing	
Selecting a Physical Architecture	Hardware and Software Specification
Infrastructure Design	Applying the Concepts at CD Selections
Deployment Diagram	Summary

INTRODUCTION

In today's environment, most information systems are spread across two or more computers. A Web-based system, for example, will run in the browser on your desktop computer, but will interact with the Web server (and possibly other computers) over the Internet. A

system that operates completely inside a company's network may have a Visual Basic program installed on your computer but interact with a database server elsewhere on the network. Therefore, an important step of design is the creation of the physical architecture layer design, the plan for how the system will be distributed across the computers and what hardware and software will be used for each computer (e.g., Windows, Linux).

Most systems are built to use the existing hardware and software in the organization, so often the current architecture and hardware and software infrastructure restricts the choice. Other factors, such as corporate standards, existing site licensing agreements, and product–vendor relationships also can mandate what architecture, hardware, and software the project team must design. However, many organizations now have a variety of infrastructures available or are openly looking for pilot projects to test new architectures, hardware, and software, which enables a project team to select an architecture on the basis of other important factors.

Designing a physical architecture layer can be quite difficult; therefore, many organizations hire expert consultants or assign very experienced analysts to the task.¹ In this chapter, we will examine the key factors in physical architecture layer design, but it is important to remember that it takes lots of experience to do it well. The nonfunctional requirements developed in early during analysis (see Chapter 5) play a key role in physical architecture layer design. These requirements are reexamined and refined into more detailed requirements that influence the system's architecture. In this chapter, we first explain the how designers think about application architectures, and we describe the three primary architectures: server-based, client-based, and client–server. Next, we look at using UML's deployment diagram as a way to model the physical architecture layer. Then we examine how the very general nonfunctional requirements from analysis are refined into more specific requirements and the implications that they have for physical architecture layer design. Finally, we consider how the requirements and architecture can be used to develop the hardware and software specifications that define exactly what hardware and other software (e.g., database systems) are needed to support the information system being developed.

ELEMENTS OF THE PHYSICAL ARCHITECTURE LAYER

The objective of designing the physical architecture layer is to determine what parts of the application software will be assigned to what hardware. In this section we first discuss the major architectural elements to understand how the software can be divided into different parts. Then we briefly discuss the major types of hardware onto which the software can be placed. Although there are numerous ways in which the software components can be placed on the hardware components, there are three principal application architectures in use today: server-based architectures, client-based architectures and client–server architectures. The most common architecture is the client–server architecture, so we focus on it.

Architectural Components

The major *architectural components* of any system are the software and the hardware. The major software components of the system being developed have to be identified and then allocated to the various hardware components on which the system will operate. Each of these components can be combined in a variety of different ways.

¹ For more information on physical architecture layer design, see Stephen D. Burd, *Systems Architecture*, 4th Ed. (Boston: Course Technology, 2003), Irv Englander, *The Architecture of Computer Hardware and Systems Software: An Information Technology Approach*, 3rd Ed. (New York: Wiley, 2003), and William Stallings, *Computer Organization & Architecture: Designing for Performance* (Upper Saddle River, NJ: Prentice Hall, 2003).

All software systems can be divided into four basic functions. The first is *data storage* (associated with the object persistence located on the data management layer—see Chapter 11). Most application programs require data to be stored and retrieved, whether the information is a small file such as a memo produced by a word processor, or a large database that stores an organization's accounting records. These are the data documented in the structural model (CRC cards and class diagrams). The second function is *data access logic* (associated with the data access and manipulation classes located on the data management layer—see Chapter 11), the processing required to access data, which often means database queries in *Structured Query Language (SQL)*. The third function is the *application logic* (located on the problem domain layer—see Chapters 6 to 10), which can be simple or complex depending on the application. This is the logic that is documented in the functional (activity diagrams and use cases) and behavioral models (sequence, communication, and behavioral state machines). The fourth function is the *presentation logic* (located on the human computer interaction layer—see Chapter 12), the presentation of information to the user, and the acceptance of the user's commands (the user interface). These four functions (data storage, data access logic, application logic, and presentation logic) are the basic building blocks of any application.

The three primary hardware components of a system are *client computers*, *servers* and the *network* that connects them. Client computers are the input/output devices employed by the user, and are usually desktop or laptop computers, but can also be handheld devices, cell phones, special-purpose terminals, and so on. Servers are typically larger computers that are used to store software and hardware that can be accessed by anyone who has permission. Servers can come in several flavors: *mainframes* (very large, powerful computers usually costing millions of dollars), *minicomputers* (large computers costing hundreds of thousands of dollars), and *microcomputers* (small desktop computers like the one you use to ones costing \$50,000 or more). The network that connects the computers can vary in speed from a slow cell phone or modem connection that must be dialed, to medium speed always-on frame relay networks, to fast always-on broadband connections such as cable modem, DSL, or T1 circuits, to high speed always-on Ethernet, T3, or ATM circuits.²

Server-Based Architectures

The very first computing architectures were *server-based architectures*, with the server (usually a central mainframe computer) performing all four application functions. The clients (usually terminals) enabled users to send and receive messages to and from the server computer. The clients merely captured keystrokes and sent them to the server for processing, and accepted instructions from the server on what to display (see Figure 13-1).

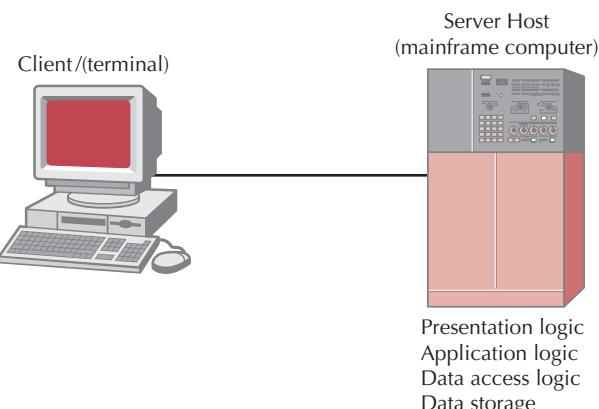


FIGURE 13-1
Server-Based
Architecture

² For more information on networks, see Alan Dennis, *Networking in the Internet Age* (New York: Wiley, 2002).

This very simple architecture often works very well. Application software is developed and stored on one computer, and all data are on the same computer. There is one point of control, because all messages flow through the one central server. The fundamental problem with server-based networks is that the server must process all messages. As the demands for more and more applications grow, many server computers become overloaded and unable to quickly process all the users' demands. Response time becomes slower, and network managers are required to spend increasingly more money to upgrade the server computer. Unfortunately, upgrades come in large increments and are expensive; it is difficult to upgrade "a little."

Client-Based Architectures

With *client-based architectures*, the clients are personal computers on a local area network, and the server computer is a server on the same network. The application software on the client computers is responsible for the presentation logic, the application logic, and the data access logic; the server simply stores the data (see Figure 13-2).

This simple architecture also often works well. However, as the demands for more and more network applications grow, the network circuits can become overloaded. The fundamental problem in client-based networks is that all data on the server must travel to the client for processing. For example, suppose the user wishes to display a list of all employees with company life insurance. All the data in the database must travel from the server where the database is stored over the network to the client, which then examines each record to see if it matches the data requested by the user. This can overload both the network and the power of the client computers.

Client–Server Architectures

Most organizations today are moving to *client–server architectures*, which attempt to balance the processing between the client and the server by having both do some of the application functions. In these architectures, the client is responsible for the presentation logic while the server is responsible for the data access logic and data storage. The application logic may reside on either the client, the server, or be split between both (see Figure 13-3). The client shown in Figure 13-3 can be referred to as a *thick* or *fat client* if it contains the bulk of application logic. A current practice is to create client–server architectures using *thin clients* because there is less overhead and maintenance in supporting thin-client applications. For example, many Web-based systems are designed with the Web browser performing presentation, with only minimal application logic using programming languages like JavaScript, and the Web server having the application logic, data access logic, and data storage.

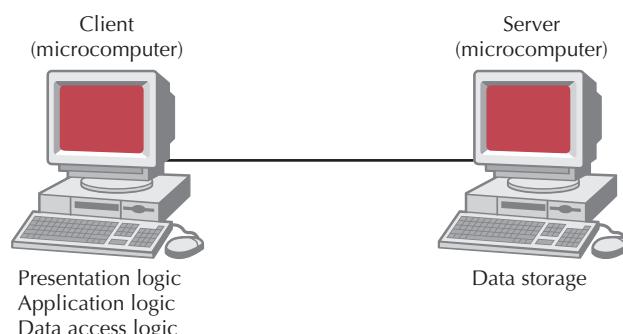


FIGURE 13-2
Client-based
Architectures

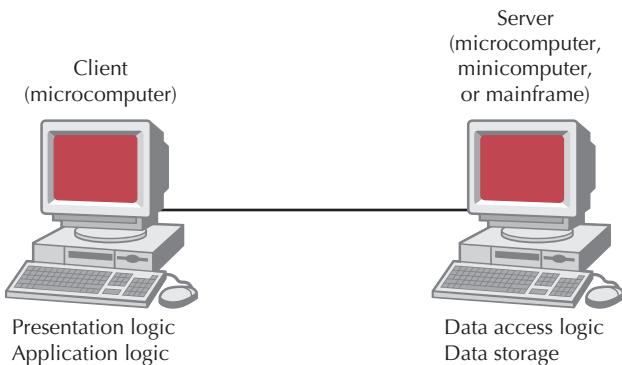


FIGURE 13-3
Client-Server
Architecture

Client-server architectures have four important benefits. First and foremost, they are *scalable*. That means it is easy to increase or decrease the storage and processing capabilities of the servers. If one server becomes overloaded, you simply add another server so that many servers are used to perform the application logic, data access logic, or data storage. The cost to upgrade is much more gradual, and you can upgrade in smaller steps rather than spending hundreds of thousands to upgrade a mainframe server.

Second, client-server architectures can support many different types of clients and servers. It is possible to connect computers that use different operating systems so that users can choose which type of computer they prefer (e.g., combining both Windows computers and Apple Macintoshes on the same network). You are not locked into one vendor, as is often the case with server-based networks. *Middleware* is a type of system software designed to translate between different vendors' software. Middleware is installed on both the client computer and the server computer. The client software communicates with the middleware that can reformat the message into a standard language that can be understood by the middleware that assists the server software.

Third, for thin client server architectures that use Internet standards, is it simple to clearly separate the presentation logic, the application logic, and the data access logic and design each to be somewhat independent. For example, the presentation logic can be designed in HTML or XML to specify how the page will appear on the screen (e.g., the colors, fonts, order of items, specific words used, command buttons, the type of selection lists and so on; see Chapter 12). Simple program statements are used to link parts of the interface to specific application logic modules that perform various functions. These HTML or XML files defining the interface can be changed without affecting the application logic. Likewise, it is possible to change the application logic without changing the presentation logic or the data, which are stored in databases and accessed using SQL commands.

Finally, because no single server computer supports all the applications, the network is generally more reliable. There is no central point of failure that will halt the entire network if it fails, as there is in a server-based computing. If any one server fails in a client-server environment, the network can continue to function using all the other servers (but, of course, any applications that require the failed server will not work).

Client-server architectures also have some critical limitations, the most important of which is its complexity. All applications in client-server computing have two parts, the software on the client and the software on the server. Writing this software is more complicated than writing the traditional all-in-one software used in server-based architectures. Updating the network with a new version of the software is more complicated, too. In server-based architectures, there is one place in which application software is stored; to update the

software, you simply replace it there. With client–server architectures, you must update all clients and all servers.

Much of the debate about server-based versus client–server architectures has centered on cost. One of the great claims of server-based networks in the 1980s was that they provided economies of scale. Manufacturers of big mainframes claimed it was cheaper to provide computer services on one big mainframe than on a set of smaller computers. The personal computer revolution changed this. Since the 1980s, the cost of personal computers has continued to drop, whereas, their performance has increased significantly. Today, personal computer hardware is more than *1,000 times cheaper* than mainframe hardware for the same amount of computing power.

With cost differences like these, it is easy to see why there has been a sudden rush to microcomputer-based client–server computing. The problem with these cost comparisons is that they ignore the *total cost of ownership*, which includes factors other than obvious hardware and software costs. For example, many cost comparisons overlook the increased complexity associated with developing application software for client–server networks. Most experts believe that it costs four to five times more to develop and maintain application software for client–server computing than it does for server-based computing.

Client–Server Tiers

There are many ways in which the application logic can be partitioned between the client and the server. The example in Figure 13-3 is one of the most common. In this case, the server is responsible for the data, and the client is responsible for the application and presentation. This is called a *two-tiered architecture* because it uses only two sets of computers, clients and servers.

A *three-tiered architecture* uses three sets of computers (see Figure 13-4). In this case, the software on the client computer is responsible for presentation logic, an application server(s) is responsible for the application logic, and a separate database server(s) is responsible for the data access logic and data storage.

An *n-tiered architecture* uses more than three sets of computers. In this case, the client is responsible for presentation, a database server(s) is responsible for the data access logic and data storage, and the application logic is spread across two or more different sets of servers. Figure 13-5 shows an example of an *n*-tiered architecture of a software product called Consensus @nyWARE®.³ Consensus @nyWARE® has four major components. The first is the Web browser on the client computer employed by a user to access the system and enter commands (presentation logic). The second component is a Web server that

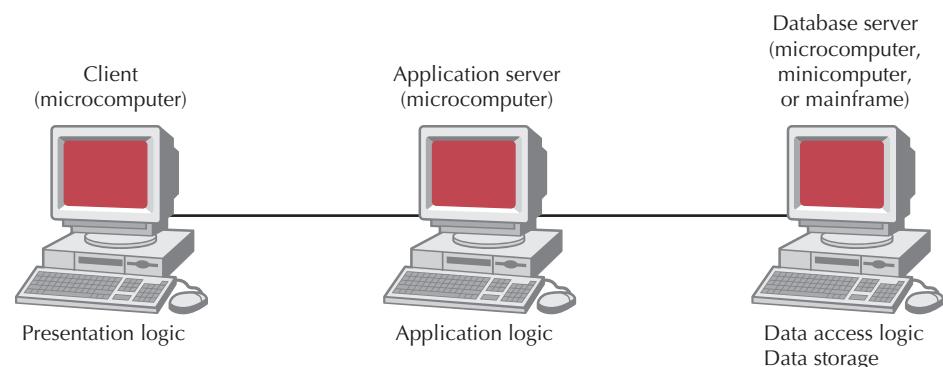


FIGURE 13-4
A Three-Tier Client-Server Architecture

³ Consensus @nyWARE® was originally developed by Alan Dennis while at the University of Georgia. It is on the Web at www.softbicycle.com.

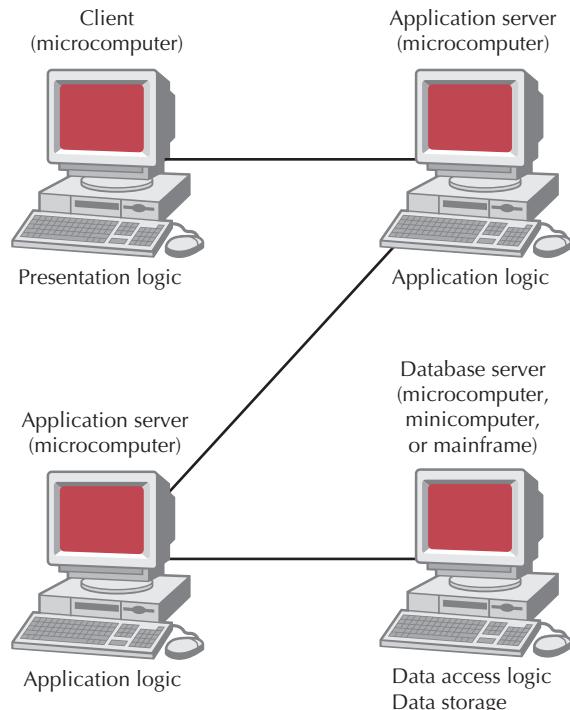


FIGURE 13-5
A Four-Tier Client-Server Architecture

responds to the user's requests, either by providing (HTML) pages and graphics (application logic) or by sending the request to the third component (a set of twenty-eight programs written in the C programming language) on another application server that performs various functions (application logic). The fourth component is a database server that stores all the data (data access logic and data storage). Each of these four components is separate, making it easy to spread the different components on different servers and to partition the application logic on two different servers.

The primary advantage of an *n*-tiered client-server architecture compared with a two-tiered architecture (or a three-tiered with a two-tiered) is that it separates out the processing that occurs to better balance the load on the different servers; it is more scalable. In Figure 13-5, we have three separate servers, a configuration that provides more power than if we had used a two-tiered architecture with only one server. If we discover that the application server is too heavily loaded, we can simply replace it with a more powerful server, or just put in several more application servers. Conversely, if we discover the database server is underused, we could store data from another application on it.

There are two primary disadvantages to an *n*-tiered architecture compared with a two-tiered architecture (or a three-tiered with a two-tiered). First, the configuration puts a greater load on the network. If you compare Figures 13-3, 13-4, and 13-5, you will see that the *n*-tiered model requires more communication among the servers; it generates more network traffic, so you need a higher-capacity network. Second, it is much more difficult to program and test software in *n*-tier architectures than in two-tiered architectures because more devices have to communicate to complete a user's transaction.

Distributed Objects Computing

From an object-oriented perspective, *distributed objects computing (DOC)* is the next version of client-server computing. DOC represents a software layer that goes between the

CONCEPTS

IN ACTION

13-A A Monster Client–Server Architecture

Every spring, Monster.com, one of the largest job sites in the United States with an average of more than 3 million visitors per month, experiences a large increase in traffic. Aaron Braham, vice president of operations, attributes the spike to college students who increase their job search activities as they approach graduation.

Monster.com uses a three-tier client–server architecture that has 150 Web servers and 30 database servers in its main site in Indianapolis. It plans to move that to 400 over the next year by gradually growing the main site and adding a new site with servers in Maynard, Massachusetts, just in time for the spring rush. The main Web site has a set of load-balancing devices that forward Web requests to the different servers depending on how busy they are.

Braham says the major challenge is that 90 percent of the traffic is not simple requests for Web pages, but rather, search requests (e.g., what network jobs are available in New Mexico), which require more processing and

access to the database servers. Monster.com has more than 350,000 job postings and more than 3 million resumes on file, spread across its database servers. Several copies of each posting and resume are kept on several database servers to improve access speed and provide redundancy in case a server crashes, so just keeping the database servers in sync so that they contain correct data is a challenge.

Questions:

1. What are the primary two or three nonfunctional requirements that have influenced Monster.com's application architecture?
2. What alternatives do you think Monster.com considered?

Source: "Resume Influx Tests Mettle of Job Sites' Scalability," *Internet Week* (May 29, 2000).

clients and servers; hence, it is known as middleware. Middleware supports the interaction between objects in a distributed computing environment. Furthermore, middleware treats the actual physical network architecture in a transparent manner. This allows the developer to focus on the development of the application and ignore the peculiarities of a specific distributed environment.

From a practical perspective, DOC allows the developer to simply concentrate on the users, objects, and methods of an application instead of worrying about which server contains which set of objects. The client object simply requests the "network" to locate and execute the server object's method. This basically allows the physical location of the server object to become irrelevant from the client object's perspective. Therefore, since servers are no longer being addressed directly on the client, servers can be added and subtracted from the network without having to update the client code. Only the middleware must be made aware of the new addresses of the server objects. This can greatly reduce the maintenance in a client–server environment. However, since middleware adds an additional layer between the clients and servers, it can reduce the efficiency of the application.

Currently, there are three competing approaches to support DOC: Object Management Group's, Sun's, and Microsoft's. The Object Management Group supports DOC via its *Common Object Request Broker Architecture* (CORBA) standard. Sun supports DOC via its *Enterprise JavaBeans* (EJB) and its *Java 2 Enterprise Edition* (J2EE). At this point in time, the approaches supported by OMG and Sun seem to be evolving toward one another. Even though Microsoft is involved in OMG, Microsoft has its own competing approach to support DOC: the *Distributed Component Object Model* (DCOM) and now, their .net initiative. At this point in time, all three are competitors to support DOC; it is not clear which approach will win in the marketplace.

Selecting a Physical Architecture

Most systems are built to use the existing infrastructure in the organization, so often the current infrastructure restricts the choice of architecture. For example, if the new system will be built for a mainframe-centric organization, a server-based architecture may be the best option. Other factors like corporate standards, existing licensing agreements, and product/vendor relationships also can mandate what architecture the project team needs to design. However, many organizations now have a variety of infrastructures available or are openly looking for pilot projects to test new architectures and infrastructures, enabling a project team to select an architecture based on other important factors.

Each of the computing architectures just discussed has its strengths and weaknesses, and no architecture is inherently better than the others. Thus, it is important to understand the strengths and weaknesses of each computing architecture and when to use each. Figure 13-6 presents a summary of the important characteristics of each architecture.

Cost of Infrastructure One of the strongest driving forces to client–server architectures is cost of infrastructure (the hardware, software, and networks that will support the application system). Simply put, personal computers are more than 1,000 times cheaper than mainframes for the same amount of computing power. The personal computers on our desks today have more processing power, memory, and hard disk space than the typical mainframe of the past, and the cost of the personal computers is a fraction of the cost of the mainframe.

Therefore, the cost of client–server architectures is low compared to server-based architectures that rely on mainframes. Client–server architectures also tend to be cheaper than client-based architectures because they place less of a load on networks and thus require less network capacity.

Cost of Development The cost of developing systems is an important factor when considering the financial benefits of client–server architectures. Developing application software for client–server computing is extremely complex, and most experts believe that it costs four to five times more to develop and maintain application software for client–server computing than it does for server-based computing. Developing application software for client-based architectures is usually cheaper still, because there are many GUI development tools for simple stand-alone computers that communicate with database servers (e.g., Visual Basic, Access).

The cost differential may change as more companies gain experience with client–server applications; new client–server products are developed and refined; and client–server standards mature. However, given the inherent complexity of client–server software and the need to coordinate the interactions of software on different computers there is likely to remain a cost difference.

FIGURE 13-6
Characteristics of Computing Architectures

	Server-Based	Client-Based	Client–Server
Cost of infrastructure	Very high	Medium	Low
Cost of development	Medium	Low	High
Ease of development	Low	High	Low–medium
Interface capabilities	Low	High	High
Control and security	High	Low	Medium
Scalability	Low	Medium	High

Ease of Development In most organizations today, there is a huge backlog of mainframe applications, systems that have been approved but that lack the staff to implement them. This backlog signals the difficulty in developing server-based systems. The tools for mainframe-based systems often are not user-friendly and require highly specialized skills (e.g., COBOL)—skills that new graduates often don't have and aren't interested in acquiring. In contrast, client-based and client–server architectures can rely on GUI development tools that can be intuitive and easy to use. The development of applications for these architectures can be fast and painless. Unfortunately, the applications for client–server can be very complex because they must be built for several layers of hardware (e.g., database servers, Web servers, client workstations) that need to communicate effectively with each other. Project teams often underestimate the effort involved in creating secure, efficient client–server applications.

Interface Capabilities Typically, server-based applications contain plain, character-based interfaces. For example, think about airline reservation systems like SABRE that can be quite difficult to use unless the operator is well trained on the commands and hundreds of codes that are used to navigate through the system. Today, most users of systems expect a *graphical user interface (GUI)* or a Web-based interface, which they can operate using a mouse and graphical objects (e.g., pushbuttons, drop-down lists, icons, and so on). GUI and Web development tools typically are created to support client-based or client–server applications; rarely can server-based environments support these types of applications.

Control and Security The server-based architecture was originally developed to control and secure data, and it is much easier to administer because all of the data are stored in a single location. In contrast, client–server computing requires a high degree of coordination among many components, and the chance for security holes or control problems is much more likely. Also, the hardware and software that are used in client–server are still maturing in terms of security. When an organization has a system that absolutely must be secure (e.g., an application used by the Department of Defense), then the project team may be more comfortable with the server-based alternative on highly secure and control-oriented mainframe computers.

Scalability Scalability refers to the ability to increase or decrease the capacity of the computing infrastructure in response to changing capacity needs. The most scalable architecture is client–server computing because servers can be added to (or removed from) the architecture when processing needs change. Also, the types of hardware that are used in client–server (e.g., minicomputers) typically can be upgraded at a pace that most closely matches the growth of the application. In contrast, server-based architectures rely primarily on mainframe hardware that needs to be scaled up in large, expensive increments, and client-based architectures have ceilings above which the application cannot grow because increases in use and data can result in increased network traffic to the extent that performance is unacceptable.

YOUR TURN

13-1 Course Registration System

Think about the course registration system in your university. What physical architecture does it use? If you had to create a new system today, would you use the

current architecture, or change to a different one? Describe the criteria that you would consider when making your decision.

INFRASTRUCTURE DESIGN

In most cases, a system is built for an organization that has a hardware, software, and communications infrastructure already in place. Thus, project teams are usually more concerned with how an existing infrastructure needs to be changed or improved to support the requirements that were identified during analysis, as opposed to how to design and build an infrastructure from scratch. Further, the coordination of infrastructure components is very complex, and it requires highly skilled technical professionals. As a project team, it is best to leave changes to the computing infrastructure to the infrastructure analysts. In this section we summarize key elements of infrastructure design to give you a basic understanding of what it includes. We describe UML's deployment diagram and the network model.

Deployment Diagram

Deployment diagrams are used to represent the relationships between the hardware components used in the physical infrastructure of an information system. For example, when designing a distributed information system that will use a wide area network, a deployment diagram can be used to show the communication relationships among the different nodes in the network. They also can be used to represent the software components and how they are deployed over the physical architecture or infrastructure of an information system. In this case, a deployment diagram represents the environment for the execution of the software.

The elements of a deployment diagram include nodes, artifacts, and communication paths (see Figure 13-7). There are also other elements that can be included in this diagram. In our case, we only include the three primary elements and the element that portrays an artifact being deployed onto a node.

A *node* represents any piece of hardware that needs to be included in the model of the physical architecture layer design. For example, nodes typically include client computers, servers, separate networks, or individual network devices. Typically, a node is labeled with its name and possibly with a stereotype. A stereotype is modeled as a text item enclosed within guillemets “<>” symbols. The stereotype represents the type of node being represented on the diagram. For example, typical stereotypes include device, mobile device, database server, web server, and application server. Finally, there are times that the notation of a node should be extended to better communicate the design of the physical architecture layer. Figure 13-8 includes a set of typical network node symbols that can be used instead of the standard notation.

An *artifact* represents a piece of the information system that is to be deployed onto the physical architecture (see Figure 13-7). Typically, an artifact represents a software component, a subsystem, database table, an entire database, or a layer (data management, human computer interaction, or problem domain). Artifacts, like nodes, can be labeled with both a name and a stereotype. Stereotypes for artifacts include source file, database table, and executable file.

A *communication path* represents a communication link between the nodes of the physical architecture (see Figure 13-7). Communication paths are stereotyped based on the type of communication link they represent (e.g., LAN, Internet, serial, parallel, or USB) or the protocol that is being supported by the link (e.g., TCP/IP).

Figure 13-9 portrays three different versions of a deployment diagram. Version A only uses the basic standard notation. Version B introduces the idea of deploying an artifact onto a node (see Figure 13-7). In this case, the artifacts represent the different layers of the appointment system described in the earlier chapters. Finally, Version C uses the extended notation to represent the same architecture. As you can see, all three versions have their strengths and weaknesses. When comparing Version A and Version B, the user can glean more information from Version B with little additional effort. However, when comparing Version A to Version C, the extended node notation enables the user to quickly understand the hardware requirements of the architecture.

Finally, when comparing Version B to Version C, Version B supports the software distribution explicitly, but forces the user to rely on the stereotypes to understand the required hardware, while Version C omits the software distribution information entirely. We recommend that you use the combination of symbols to best portray the physical architecture to the user community.

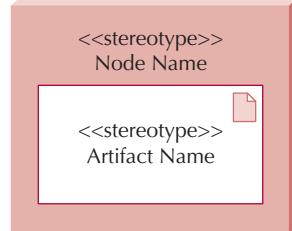
A Node:	
An Artifact:	
A Node with a Deployed Artifact:	
A Communication Path:	

FIGURE 13-7 Development Diagram Syntax

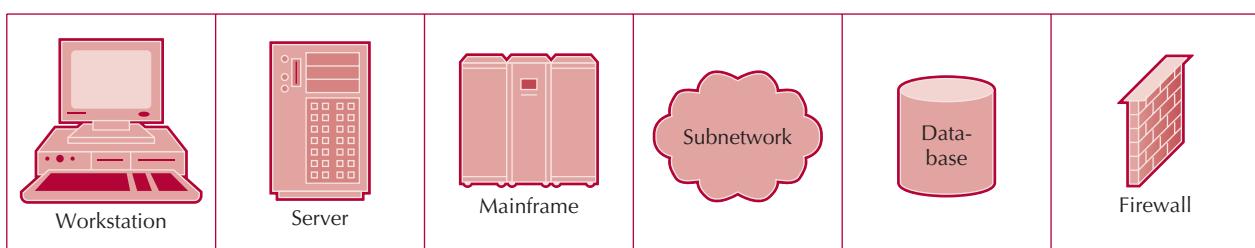


FIGURE 13-8 Extended Node Syntax for Development Diagram

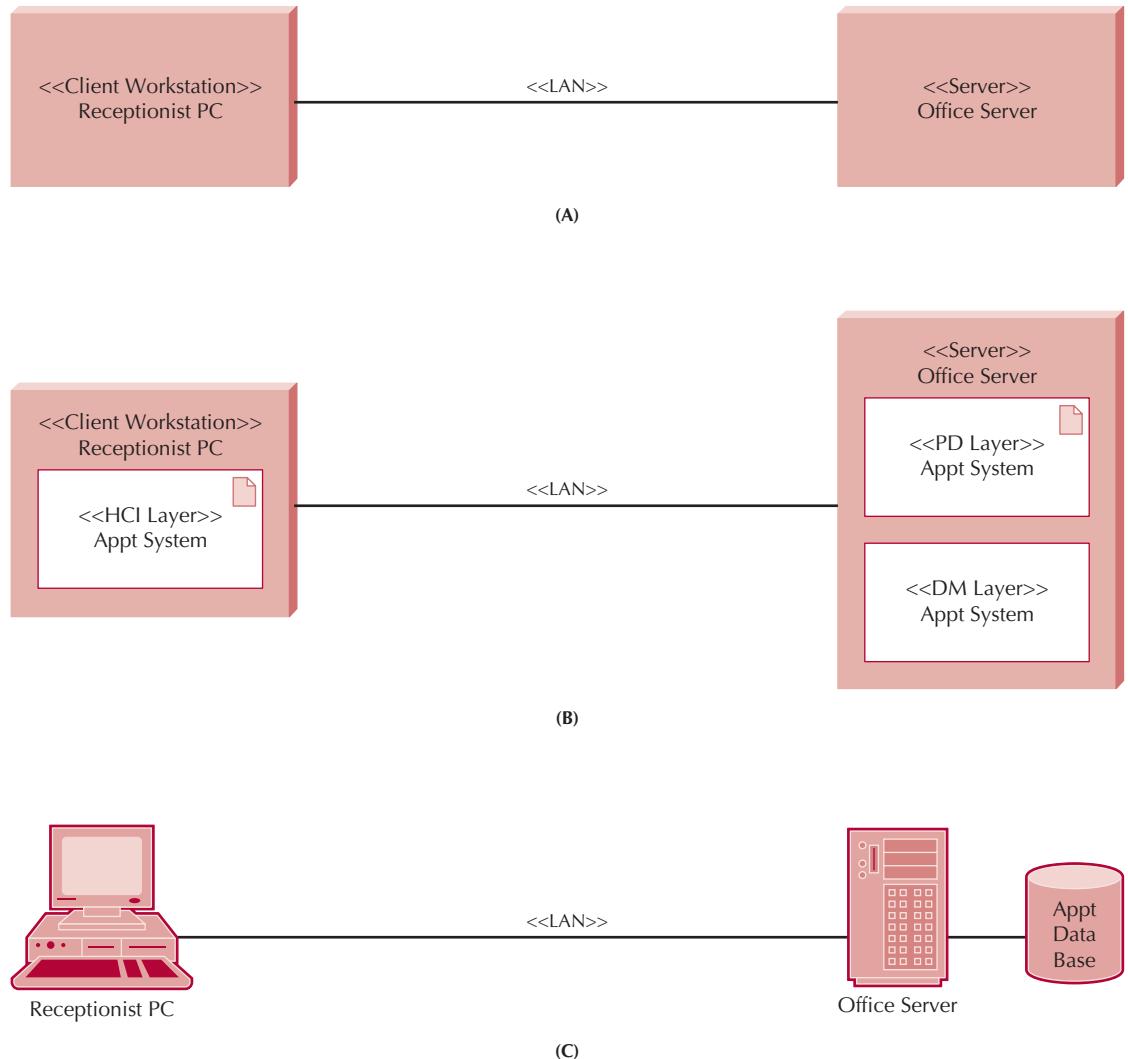


FIGURE 13-9 Three Versions of Appointment System Deployment Diagram

The Network Model

The *network model* is a diagram that shows the major components of the information system (e.g., servers, communication lines, networks) and their geographic locations throughout the organization. There is no one way to depict a network model, and in our experience analysts create their own standards and symbols, using presentation applications (e.g., Microsoft PowerPoint) or diagramming tools (e.g., Visio). In this text, we use UML's deployment diagram.

The purpose of the network model is twofold: to convey the complexity of the system and to show how the system's software components will fit together. The diagram also helps the project team develop the hardware and software specification that is described later in this chapter.

The components of the network model are the various clients (e.g., personal computers, kiosks), servers (e.g., database, network, communications, printer), network equipment (e.g., wires, dial-up connections, satellite links), and external systems or networks (e.g.,

Internet service providers) that support the application. *Locations* are the geographic sites related to these components. For example, if a company created an application for users at four of its plants in Canada and eight plants in the United States, and it used one external system to provide Internet service, the network model to depict this would contain twelve locations ($4 + 8 = 12$).

Creating the network model is a top-down exercise whereby you first graphically depict all of the locations where the application will reside. This is accomplished by placing symbols that represent the locations for the components on a diagram, and then connecting them with lines that are labeled with the approximate amount of data or types of network circuits between the separated components.

Companies seldom build networks to connect distant locations by buying land and laying cable (or sending up their own satellites). Instead, they usually lease services provided by large telecommunications firms such as AT&T, Sprint, and Verizon. Figure 13-10 shows a typical network. The “clouds” in the diagram represent the networks at different locations (e.g., Toronto, Atlanta). The lines represent network connections between specific points (e.g., Toronto to Brampton). In other cases, a company may lease connections from many points to many others, and rather than trying to show all the connections, a separate “cloud” may be drawn to represent this many-to-many type of connection (e.g., the cloud in the center of this Figure 13-10 represents a network of many-to-many connections provided by a telecom firm like Verizon).

This high-level diagram has several purposes. First, it shows the locations of the components that are needed to support the application; therefore, the project team can get a good understanding of the geographic scope of the new system and how complex and costly the communications infrastructure will be to support. (For example, an application that supports one site likely will have less communications costs as compared to a more complex application that will be shared all over the world.) The diagram also indicates the external components of the system (e.g., customer systems, supplier systems), which may impact security or global needs (discussed later in this chapter).

The second step to the network model is to create low-level network diagrams for each of the locations shown on the top-level diagram. First, hardware is drawn on the model in a way that depicts how the hardware for the new system will be placed throughout the location. It usually helps to use symbols that resemble the hardware that will be used. The amount of detail to include on the network model depends on the needs of the

CONCEPTS

IN ACTION

13-B Nortel's Network

Nortel, the giant Canadian manufacturer of network switches, recently implemented a network using its own equipment. Prior to implementing it, Nortel had used a set of more than 100 circuits that was proving to be more and more expensive and unwieldy to manage. Nortel started by identifying the 20 percent of its sites that accounted for 80 percent of the network traffic. These were the first to move to the new network. By the time the switchover is complete, more than 100 sites will be connected.

There are two primary parts to the new network (see Figure 13-10). The largest part uses a public network

provided by WorldCom/MCI. Nortel also operates a private network in Ontario. Both parts of the network use 44 Mbps circuits. Nortel is planning to move to 155 Mbps circuits when costs drop.

Question:

1. What are the advantages and disadvantages of Nortel's phased implementation of the new network?

Source: "A Case for ATM," *Network World* (June 2, 1997).

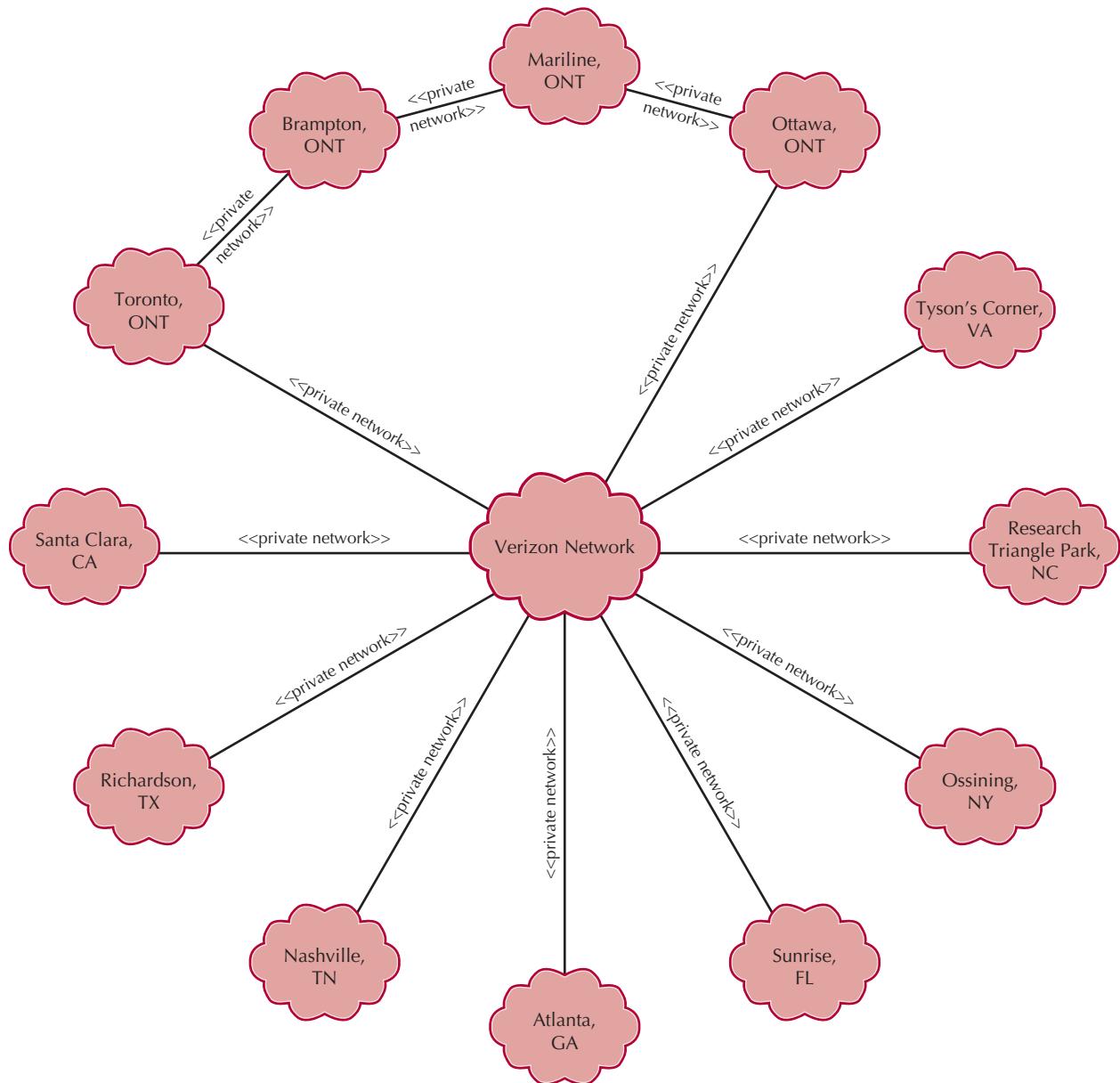


FIGURE 13-10 Deployment Diagram Representation of a Top-Level Network Model

project. Some low-level network models contain text descriptions below each of the hardware components that describe in detail the proposed hardware configurations and processing needs; others only include the number of users that are associated with the different parts of the diagram.

Next, lines are drawn connecting the components that will be physically attached to each other. In terms of software, some network models list the required software for each network model component right on the diagram; whereas, other times, the software is described in a memo that is attached to the network model and stored in the project binder. Figure 13-11 shows a deployment diagram that portrays two levels of

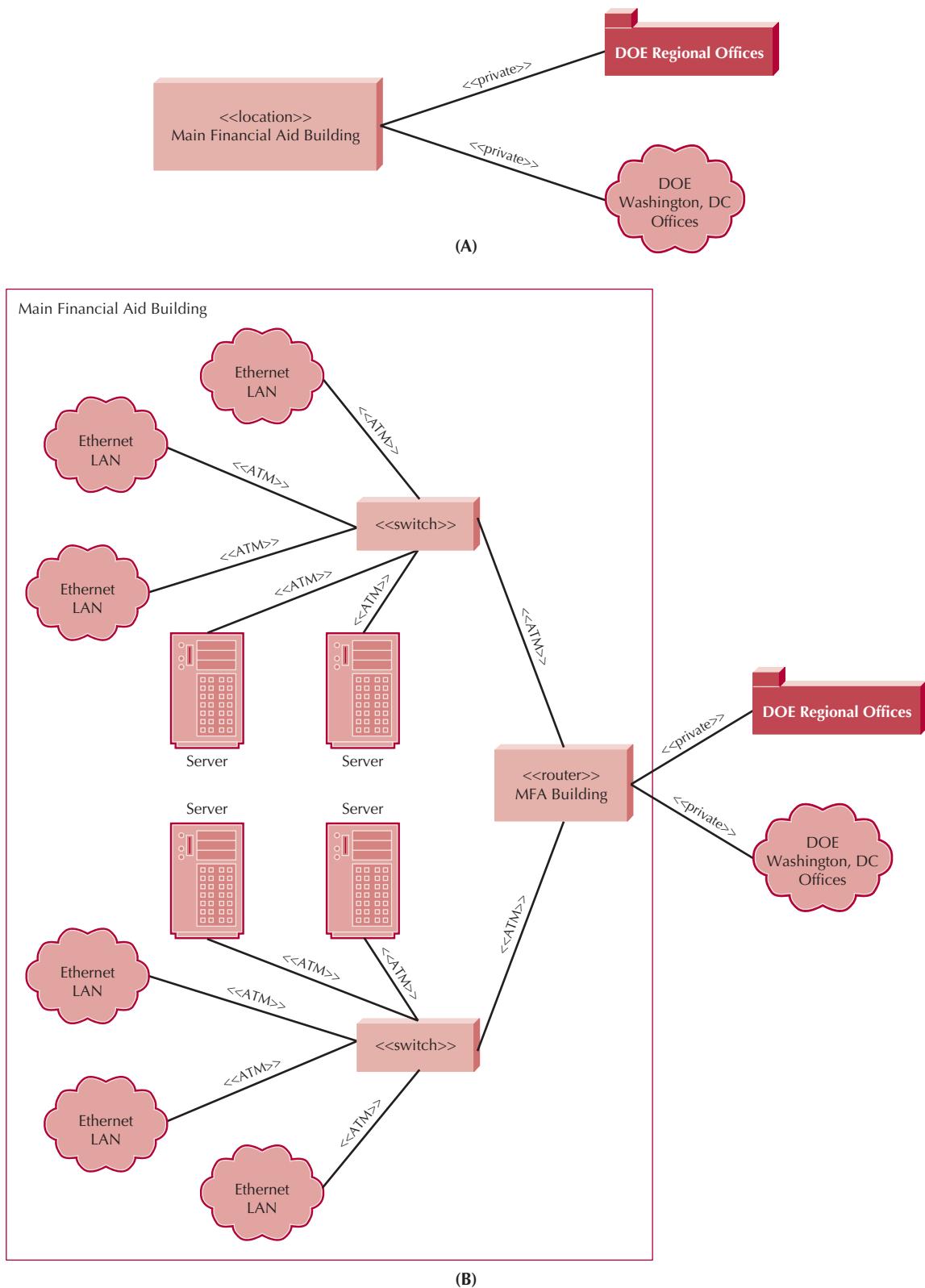


FIGURE 13-11 Deployment Diagram Representation of a Low-Level Network Model

detail of a low-level network model. Notice, we use both the standard and extended node notation in this figure. Furthermore, in this case, we have included a package (see Chapter 9) to represent a set of connections to the router in the MFA building. By including a package, we only show the detail necessary. The extended notation in many cases aids the user in understanding the topology of the physical architecture layer much better than the standard notation. As such, we recommend you use the symbols that get the message across best.

Our experiences have shown that most project teams create a memo for the project files that provides additional detail about the network model. This information is helpful to the people who are responsible for creating the hardware and software specification (described later in this chapter) and who will work more extensively with the infrastructure development. This memo can include special issues that impact communications, requirements for hardware and software that may not be obvious from the network model, or specific hardware or software vendors or products that should be acquired.

Again, the primary purpose of the network model diagram is to present the proposed infrastructure for the new system. The project team can use the diagrams to understand the scope of the system, the complexity of its structure, any important communication issues that may affect development and implementation, and the actual components that will need to be acquired or integrated into the environment.

CONCEPTS

IN ACTION

13-C The Department of Education's Financial Aid Network

Each year, the U.S. Department of Education (DOE) processes more than 10 million student loans, such as Pell Grants, Guaranteed Student Loans, and college work-study programs. The DOE has more than 5,000 employees that need access to more than 400 gigabytes of student data.

The financial aid network, located in one Washington, D.C., building, has two major parts (i.e., backbones) that each use high-speed 155 Mbps circuits to connect four local area networks and three high-performance database servers (see Figure 13-11). These two backbones are connected into another network backbone

that connects the financial aid building using 100 Mbps circuits to five other DOE buildings in Washington with similar networks. The two backbones are also connected to the ten regional DOE offices across the U.S. and to other U.S. government agencies via a set of lower speed 1.5 Mbps connections.

Source: *Network Computing* (May 15, 1996).

Question:

1. Why does each part of the network have different data rates?

YOUR TURN

13-2 Create a Network Model

Pretend that you have just moved into a fraternity/sorority house with twenty other people. The house has a laser printer and a low-cost scanner that its inhabitants are free to use. You have decided to network the house so that everyone can

share the printer and scanner and have dial-up access to the university network. Create a high-level network model that describes the locations that will be involved in your work. Next, create a low-level diagram for the house itself.

CONCEPTS

IN ACTION

13-D The Importance of Infrastructure Planning

At the end of 1997, Oxford Health Plans, Inc. posted a \$120 million loss to its books. The company's unexpected growth was its undoing. The system, which was originally planned to support the company's 217,000 members, had to meet the needs of a membership that exceeded 1.5 million. System users found that processing a new member sign-up took 15 minutes instead of the proposed 6 seconds. Also, the computer problems left Oxford unable to send out bills to many of its customer accounts and rendered it unable to track payments to hundreds of doctors and hospitals. In less than a year,

uncollected payments from customers tripled to more than \$400 million, while the payments owed to caregivers amounted to more than \$650 million. Mistakes in infrastructure planning can cost far more than the cost of hardware, software, and network equipment alone.

Source: *The Wall Street Journal* (December 11, 1997).

Question:

1. If you were in charge of the Oxford project, how would you have avoided these problems?

NONFUNCTIONAL REQUIREMENTS AND PHYSICAL ARCHITECTURE LAYER DESIGN

The design of the physical architecture layer specifies the overall architecture and the placement of software and hardware that will be used. Each of the architectures discussed above has its strengths and weaknesses. Most organizations are trying to move to client–server architectures for cost reasons, so in the event that there is no compelling reason to choose one architecture over another, cost usually suggests client–server.

Creating a physical architecture layer design begins with the nonfunctional requirements. The first step is to refine the nonfunctional requirements into more detailed requirements that are then used to help select the architecture to be used (server-based, client-based, or client–server) and what software components will be placed on each device. In a client–server architecture, one also has to decide whether to use a two-tier, three-tier, or n -tier architecture. Then the nonfunctional requirements and the architecture design are used to develop the hardware and software specification.

There are four primary types of nonfunctional requirements that can be important in designing the architecture: operational requirements, performance requirements, security requirements, and cultural/political requirements. We describe each in turn and then explain how they may affect the physical architecture layer design.

Operational Requirements

Operational requirements specify the operating environment(s) in which the system must perform and how those may change over time. This usually refers to operating systems, system software, and information systems with which the system must interact, but it will on occasion also include the physical environment if the environment is important to the application (e.g., it's located on a noisy factory floor, so no audible alerts can be heard). Figure 13-12 summarizes four key operational requirement areas and provides some examples of each.

Technical Environment Requirements *Technical environment requirements* specify the type of hardware and software system on which the system will work. These requirements usually focus on the operating system software (e.g., Windows, Linux), database system software (e.g., Oracle), and other system software (e.g., Internet Explorer). Occasionally,

Type of Requirement	Definition	Examples
Technical Environment Requirements	Special hardware, software, and network requirements imposed by business requirements	<ul style="list-style-type: none"> The system will work over the Web environment with Internet Explorer. All office locations will have an always-on network connection to enable real-time database updates. A version of the system will be provided for customers connecting over the Internet via a small screen PDA.
System Integration Requirements	The extent to which the system will operate with other systems	<ul style="list-style-type: none"> The system must be able to import and export Excel spreadsheets. The system will read and write to the main inventory database in the inventory system.
Portability Requirements	The extent to which the system will need to operate in other environments	<ul style="list-style-type: none"> The system must be able to work with different operating systems (e.g., Linux and Windows XP). The system may need to operate with handheld devices such as a Palm.
Maintainability Requirements	Expected business changes to which the system should be able to adapt	<ul style="list-style-type: none"> The system will be able to support more than one manufacturing plant with six months advance notice. New versions of the system will be released every six months.

FIGURE 13-12 Operational Requirements

specific hardware requirements will impose important limitations on the system, such as the need to operate on a PDA or cell phone with a very small display.

System Integration Requirements *System integration requirements* are those that require the system to operate with other information systems, either inside or outside the company. These typically specify interfaces through which data will be exchanged with other systems.

Portability Requirements Information systems never remain constant. Business needs change and operating technologies change, so the information systems that support them and run on them must change, too. *Portability requirements* define how the technical operating environments may change over time and how the system must respond (e.g., the system may run currently on Windows XP, while in the future it may be required to deploy the system on Linux). Portability requirements also refer to potential changes in business requirements that will drive technical environment changes. For example, in the future, users may want to access a Web site from their cell phones.

Maintainability Requirements *Maintainability requirements* specify the business requirement changes that can be anticipated. Not all changes are predictable, but some are. For example, suppose a small company has only one manufacturing plant but is anticipating the construction of a second plant in the next five years. All information systems must be written to make it easy to track each plant separately, whether for personnel, budgeting, or inventory system. The maintainability requirements attempt to anticipate future requirements so that the systems designed today will be easy to maintain if and when those future requirements appear. Maintainability requirements may also define the update cycle for the system, such as the frequency with which new versions will be released.

Performance Requirements

Performance requirements focus on performance issues, such as response time, capacity, and reliability. Figure 13-13 summarizes three key performance requirement areas and provides some examples.

Speed Requirements *Speed requirements* are exactly what they say: how fast should the system operate. First and foremost, this is the *response time* of the system: how long does it take the system to respond to a user request. Although everyone would prefer low response times with the system responding immediately to each user request, this is not practical. We could design such a system, but it would be expensive. Most users understand that certain parts of a system will respond quickly, while others are slower. Those actions that are performed locally on the user's computer must be almost immediate (e.g., typing, dragging and dropping), while others that require communicating across a network can have higher response times (e.g., a Web request). In general, response times less than seven seconds are considered acceptable when they require communication over a network.

The second aspect of speed requirements is how long it takes transactions in one part of the system to be reflected in other parts. For example, how soon after an order is placed will the items it contained be shown as no longer available for sale to someone else? If the inventory is not updated immediately, then someone else could place an order for the same item, only to find out later it is out of stock. Or, how soon after an order is placed is it sent to the warehouse to be picked from inventory and shipped? In this case, some time delay might have little impact.

Capacity Requirements *Capacity requirements* attempt to predict how many users the system will have to support, both in total and simultaneously. Capacity requirements are important in understanding the size of the databases, the processing power needed and so on. The most important requirement is usually the peak number of simultaneous users because this has a direct impact on the processing power of the computer(s) needed to support the system.

Type of Requirement	Definition	Examples
Speed Requirements	The time within which the system must perform its functions	<ul style="list-style-type: none"> Response time must be less than 7 seconds for any transaction over the network. The inventory database must be updated in real time. Orders will be transmitted to the factory floor every 30 minutes.
Capacity Requirements	The total and peak number of users and the volume of data expected	<ul style="list-style-type: none"> There will be a maximum of 100–200 simultaneous users at peak use times. A typical transaction will require the transmission of 10K of data. The system will store data on approximately 5,000 customers for a total of about 2 meg of data.
Availability and Reliability Requirements	The extent to which the system will be available to the users and the permissible failure rate due to errors	<ul style="list-style-type: none"> The system should be available 24 /7, with the exception of scheduled maintenance. Scheduled maintenance shall not exceed one 6-hour period each month. The system shall have 99% uptime performance.

FIGURE 13-13 Performance Requirements

It is often easier to predict the number of users for internal systems designed to support an organization's own employees than it is to predict the number of users for customer-facing systems, especially those on the Web. How does weather.com estimate the peak number of users who will simultaneously seek weather information? This is as much an art as a science, so often the team provides a range of estimates, with wider ranges used to signal a less accurate estimate.

Availability and Reliability Requirements *Availability and reliability requirements* focus on the extent to which users can assume that the system will be available for them to use. Although some systems are intended to be used only during the forty-hour workweek, some systems are designed to be used by people around the world. For such systems, project team members need to consider how the application can be operated, supported, and maintained, 24/7 (i.e., 24 hours a day, 7 days a week). This 24/7 requirement means that users may need help or have questions at any time, and a support desk that is available eight hours a day will not be sufficient support. It is also important to consider what reliability is needed in the system. A system that requires high reliability (e.g., a medical device or telephone switch) needs far greater planning and testing than one that does not have such high reliability needs (e.g., personnel system, Web catalog).

It is more difficult to predict the peaks and valleys in usage of the system when the system has a global audience. Typically, applications are backed up on weekends or late evenings when users are no longer accessing the system. Such maintenance activities need to be rethought with global initiatives. The development of Web interfaces in particular has escalated the need for 24/7 support; by default the Web can be accessed by anyone at any time. For example, the developers of a Web application for U.S. outdoor gear and clothing retailer Orvis were surprised when the first order after going live came from Japan.

Security Requirements⁴

Security is ability to protect the information system from disruption and data loss, whether caused by an intentional act (e.g., a hacker or a terrorist attack) or a random event (e.g., disk failure, tornado). Security is primarily the responsibility of the operations group—the staff responsible for installing and operating security controls, such as firewalls, intrusion detection systems, and routine backup and recovery operations. Nonetheless, developers of new systems must ensure that the system's *security requirements* produce reasonable precautions to prevent problems; system developers are responsible for ensuing security within the information systems themselves.

CONCEPTS

13-E The Importance of Capacity Planning

IN ACTION

Reread the Concepts in Action 13-D that describes what happened to Oxford Health Plans in 1997.

Question:

1. If you had been in charge of the Oxford project, what things would you have considered when planning the system capacity?

⁴ For more information, see Brett C. Tjaden, *Fundamentals of Secure Computer Systems* (Wilsonville, OR: Franklin, Beedle, and Associates, 2004).

Security is an ever-increasing problem in today's Internet-enabled world. Historically, the greatest security threat has come from inside the organization itself. Ever since the early 1980s when the FBI first began keeping computer crime statistics and security firms began conducting surveys of computer crime, organizational employees have perpetrated the vast majority of computer crimes. For years, 80 percent of unauthorized break-ins, thefts, and sabotage have been committed by insiders, leaving only 20 percent to hackers external to the organizations.

In 2001, that changed. Depending on what survey you read, the percent of incidents attributed to external hackers in 2001 increased to 50 to 70 percent of all incidents, meaning that the greatest risk facing organizations is now from the outside. Although some of this shift may be due to better internal security and better communications with employees to prevent security problems, much of it is simply due to an increase in activity by external hackers.

Developing security requirements usually starts with some assessment of the value of the system and its data. This helps pinpoint extremely important systems so that the operations staff are aware of the risks. Security within systems usually focuses on specifying who can access what data, identifying the need for encryption and authentication, and ensuring the application prevents the spread of viruses (see Figure 13-14).

System Value The most important computer asset in any organization is not the equipment; it is the organization's data. For example, suppose someone destroyed a mainframe computer worth \$10 million. The mainframe could be replaced, simply by buying a new one. It would be expensive, but the problem would be solved in a few weeks. Now suppose someone destroyed all the student records at your university so that no one knew what courses anyone had taken or their grades. The cost would far exceed the cost of replacing a \$10 million computer. The lawsuits alone would easily exceed \$10 million, and the cost of staff to find and re-enter paper records would be enormous and certainly would take more than a few weeks.

In some cases, the information system itself has value that far exceeds the cost of the equipment as well. For example, for an Internet bank that has no brick and mortar branches, the Web site is a *mission critical system*. If the Web site crashes, the bank cannot conduct business with its customers. A mission critical application is an information system that is

Type of Requirement	Definition	Examples
System Value Estimates	Estimated business value of the system and its data	<ul style="list-style-type: none"> The system is not mission critical but a system outage is estimated to cost \$50,000 per hour in lost revenue. A complete loss of all system data is estimated to cost \$20 million.
Access Control Requirements	Limitations on who can access what data	<ul style="list-style-type: none"> Only department managers will be able to change inventory items within their own department. Telephone operators will be able to read and create items in the customer file, but cannot change or delete items.
Encryption and Authentication Requirements	Defines what data will be encrypted where and whether authentication will be needed for user access	<ul style="list-style-type: none"> Data will be encrypted from the user's computer to the Web site to provide secure ordering. Users logging in from outside the office will be required to authenticate.
Virus Control Requirements	Requirements to control the spread of viruses	<ul style="list-style-type: none"> All uploaded files will be checked for viruses before being saved in the system.

FIGURE 13-14 Security Requirements

literally critical to the survival of the organization. It is an application that cannot be permitted to fail, and if it does fail the network staff drops everything else to fix it. Mission critical applications are usually clearly identified so their importance is not overlooked.

Even temporary disruptions in service can have significant costs. The cost of disruptions to a company's primary Web site or the LANs and backbones that support telephone sales operations are often measured in the millions. Amazon.com, for example, has revenues of more than \$10 million per hour, so if their Web site were unavailable for an hour or even part of an hour, it would cost them millions of dollars in lost revenue. Companies that do less e-business or telephone sales have lower costs, but recent surveys suggest losses of \$100,000 to \$200,000 per hour are not uncommon for major customer-facing information systems.

Access Control Requirements Some of the data stored in the system need to be kept confidential; some data need special controls on who is allowed to change or delete it. Personnel records, for example, should only be able to be read by the personnel department and the employee's supervisor; changes should only be permitted to be made by the personnel department. *Access control requirements* state who can access what data and what type of access is permitted: whether the individual can create, read, update and/or delete the data. The requirements reduce the chance that an authorized user of the system can perform unauthorized actions.

Encryption and Authentication Requirements One of the best ways to prevent unauthorized access to data is *encryption*, which is a means of disguising information by the use of mathematical algorithms (or formulas). Encryption can be used to protect data stored in databases or data that are in transit over a network from a database to a computer. There are two fundamentally different types of encryption: symmetric and asymmetric. A *symmetric encryption algorithm* (such as Data Encryption Standard [DES] or Advanced Encryption Standard [AES]) is one in which the key used to encrypt a message is the *same* as the one used to decrypt it, which means that it is essential to protect the key and that a separate key must be used for each person or organization with whom the system shares information (or else everyone can read all the data).

An *asymmetric encryption algorithm* (such as *public key encryption*) is one in which the key used to encrypt data (called the *public key*) is different from the one used to decrypt it (called the *private key*). Even if everyone knows the public key, once the data are encrypted, they cannot be decrypted without the private key. Public key encryption greatly reduces the

CONCEPTS

13-F Power Outage Costs a Million Dollars

IN ACTION

Lithonia Lighting, located just outside of Atlanta, is the world's largest manufacturer of light fixtures with more than \$1 billion in annual sales. One afternoon, the power transformer at its corporate headquarters exploded, leaving the entire office complex, including the corporate data center, without power. The data center's backup power system immediately took over and kept critical parts of the data center operational. However, it was insufficient to power all systems, so the system supporting sales for all of Lithonia Lighting's North American agents, dealers, and distributors had to be turned off.

The transformer was quickly replaced and power was restored. However, the three-hour shutdown of the sales system cost \$1 million in potential sales lost. Unfortunately, it is not uncommon for the cost of a disruption to be hundreds or thousands of times the cost of the failed components.

Question:

- What would you recommend to avoid similar losses in the future?

key management problem. Each user has its public key that is used to encrypt messages sent to it. These public keys are widely publicized (e.g., listed in a telephone book-style directory)—that's why they're called “public” keys. The private key, in contrast, is kept secret (and why it's called “private”).

Public key encryption also permits *authentication* (or digital signatures). When one user sends a message to another, it is difficult to legally prove who actually sent the message. Legal proof is important in many communications, such as bank transfers and buy/sell orders in currency and stock trading, which normally require legal signatures. Public key encryption algorithms are *invertible*, meaning that text encrypted with either key can be decrypted by the other. Normally, we encrypt with the public key and decrypt with the private key. However, it is possible to do the inverse: encrypt with the private key and decrypt with the public key. Since the private key is secret, only the real user could use it to encrypt a message. Thus, a digital signature or authentication sequence is used as a legal signature on many financial transactions. This signature is usually the name of the signing party plus other unique information from the message (e.g., date, time, or dollar amount). This signature and the other information are encrypted by the sender using the private key. The receiver uses the sender's public key to decrypt the signature block and compares the result to the name and other key contents in the rest of the message to ensure a match.

The only problem with this approach lies in ensuring that the person or organization that sent the document with the correct private key is the actual person or organization. Anyone can post a public key on the Internet, so there is no way of knowing for sure who they actually are. For example, it would be possible for someone other than “Organization A” in this example to claim to be Organization A when in fact they are an imposter.

This is where the Internet's public key infrastructure (PKI) becomes important.⁵ The PKI is a set of hardware, software, organizations, and policies designed to make public key encryption work on the Internet. PKI begins with a *certificate authority* (CA), which is a trusted organization that can vouch for the authenticity of the person or organization using authentication (e.g., VeriSign). A person wanting to use a CA registers with the CA and must provide some proof of identify. There are several levels of certification, ranging from a simple confirmation from a valid e-mail address to a complete police-style background check with an in-person interview. The CA issues a digital certificate that is the requestor's public key encrypted using the CA's private key as proof of identify. This certificate is then attached to the user's e-mail or Web transactions in addition to the authentication information. The receiver then verifies the certificate by decrypting it with the CA's public key—and must also contact the CA to ensure that the user's certificate has not been revoked by the CA.

The *encryption and authentication requirements* state what encryption and authentication requirements are needed for what data. For example, will sensitive data such as customer credit card numbers be stored in the database in encrypted form, or will encryption be used to take orders over the Internet from the company's Web site? Will users be required to use a digital certificate in addition to a standard password?

Virus Control Requirements *Virus control requirements* address the single most common security problem: *viruses*. Recent studies have shown that almost 90 percent of organizations suffer a virus infection each year. Viruses cause unwanted events—some harmless (such as nuisance messages), some serious (such as the destruction of data). Any time a system permits data to be imported or uploaded from a user's computer, there is the potential for a virus infection. Many systems require that all information system systems that permit the import or upload of user files to check those files for viruses before they are stored in the system.

⁵ For more on the PKI, see www.ietf.org/internet-drafts/draft-ietf-pkix-roadmap-06.txt.

Cultural and Political Requirements

Cultural and political requirements are those that are specific to the countries in which the system will be used. In today's global business environment, organizations are expanding their systems to reach users around the world. Although this can make great business sense, its impact on application development should not be underestimated. Yet another important part of the design of the system's physical architecture is understanding the global cultural and political requirements for the system (see Figure 13-15).

Multilingual Requirements The first and most obvious difference between applications used in one region and those designed for global use is language. Global applications often have *multilingual requirements*, which means that they have to support users who speak different languages and write using non-English letters (e.g., those with accents, Cyrillic, Japanese). One of the most challenging aspects in designing global systems is getting a good translation of the original language messages into a new language. Words often have similar meanings but can convey subtly different meanings when they are translated, so it is important to use translators skilled in translating technical words.

The other challenge is often screen space. In general, English-language messages usually take 20 percent to 30 percent fewer letters than their French or Spanish counterparts. Designing global systems requires allocating more screen space to messages than might be used in the English-language version.

Some systems are designed to handle multiple languages on the fly so that users in different countries can use different languages concurrently; that is, the same system supports several different languages simultaneously (a concurrent multilingual system). Other systems contain separate parts that are written in each language and must be reinstalled before a specific language can be used; that is, each language is provided by a different version of the system so that any one installation will use only one language

Type of Requirement	Definition	Examples
Multilingual Requirements	The language in which the system will need to operate	<ul style="list-style-type: none"> The system will operate in English, French, and Spanish.
Customization Requirements	Specification of what aspects of the system can be changed by local users	<ul style="list-style-type: none"> Country managers will be able to define new fields in the product database to capture country-specific information. Country managers will be able to change the format of the telephone number field in the customer database.
Making Unstated Norms Explicit	Explicitly stating assumptions that differ from country to country	<ul style="list-style-type: none"> All date fields will be explicitly identified as using the month-day-year format. All weight fields will be explicitly identified as being stated in kilograms.
Legal Requirements	The laws and regulations that impose requirements on the system	<ul style="list-style-type: none"> Personal information about customers cannot be transferred out of European Union countries into the United States. It is against U.S. federal law to divulge information on who rented what videotape, so access to a customer's rental history is permitted only to regional managers.

FIGURE 13-15 Cultural and Political Requirements

(i.e., a discrete multilingual system). Either approach can be effective, but this functionality must be designed into the system well in advance of implementation.

Customization Requirements For global applications, the project team will need to give some thought to *customization requirements*: how much of the application will be controlled by a central group and how much of the application will be managed locally. For example, some companies allow subsidiaries in some countries to customize the application by omitting or adding certain features. This decision has tradeoffs between flexibility and control because customization often makes it more difficult for the project team to create and maintain the application. It also means that training can differ between different parts of the organization, and customization can create problems when staff move from one location to another.

Unstated Norms Many countries have *unstated norms* that are not shared internationally. It is important for the application designer to make these assumptions explicit because they can lead to confusion otherwise. In the United States, the unstated norm for entering a date is the date format MM/DD/YYYY; however, in Canada and most European countries, the unstated norm is DD/MM/YYYY. Another example is the order of a person's name; does the family name come first or last? The answer depends on the locality. When you are designing global systems, it is critical to recognize these unstated norms and make them explicit so that users in different countries do not become confused. Currency is the other item sometimes overlooked in system design; global application systems must specify the currency in which information is being entered and reported.

Legal Requirements *Legal requirements* are those requirements imposed by laws and government regulations. System developers sometimes forget to think about legal regulations, but unfortunately, doing so comes at some risk because ignorance of the law is no

CONCEPTS

13-G Developing Multilingual Systems

IN ACTION

I've had the opportunity to develop two multilingual systems. The first was a special-purpose decision support system to help schedule orders in paper mills called BCW-Trim. The system was installed by several dozen paper mills in Canada and the United States, and it was designed to work in either English or French. All messages were stored in separate files (one set English, one set French), with the program written to use variables initialized either to the English or French text. The appropriate language files were included when the system was compiled to produce either the French or English version.

The second program was a groupware system called GroupSystems, for which I designed several modules. The system has been translated into dozens of different languages, including French, Spanish, Portuguese, German, Finnish, and Croatian. This system enables the user to switch between languages at will by storing messages in

simple text files. This design is far more flexible because each individual installation can revise the messages at will. Without this approach, it is unlikely that there would have been sufficient demand to warrant the development of versions to support less commonly used languages (e.g., Croatian).

—Alan Dennis

Question:

1. How would you decide how much to support users who speak languages other than English?
2. Would you create multilingual capabilities for any application that would be available to non-English speaking people? Think about Web sites that exist today.

defense. For example, in 1997, a French court convicted the Georgia Institute of Technology of violating French language law. Georgia Tech operated a small campus in France that offered summer programs for American students. The information on the campus Web server was primarily in English because classes are conducted in English, which violated the law requiring French to be the predominant language on all Internet servers in France. By formally considering legal regulations, they are less likely to be overlooked.

Synopsis

In many cases, the technical environment requirements as driven by the business requirements may simply define the physical architecture layer. In this case, the choice is simple: business requirements dominate other considerations. For example, the business requirements may specify that the system needs to work over the Web using the customer's Web browser. In this case, the architecture probably should be a thin client-server. Such business requirements are most likely in systems designed to support external customers. Internal systems may also impose business requirements, but usually they are not as restrictive.

In the event that the technical environment requirements do not require the choice of a specific architecture, then the other nonfunctional requirements become important. Even in cases when the business requirements drive the architecture, it is still important to work through and refine the remaining nonfunctional requirements because they are important in later stages of the design and implementation phases. Figure 13-16 summarizes the relationship between requirements and recommended architectures.

Operational Requirements System integration requirements may lead to one architecture over another, depending on the architecture and design of the system(s) with which the system needs to integrate. For example, if the system must integrate with a desktop system (e.g., Excel) then this may suggest a thin or thick client-server architecture, while if it must integrate with a server-based system then a server-based architecture may be indicated. Systems that have extensive portability requirements tend to be best suited for a thin client-server architecture because it is simpler to write for Web-based standards (e.g., HTML, XML) that extend the reach of the system to other platforms, rather than trying to write and rewrite extensive presentation logic for different platforms in the server-based, client-based, or thick client-server architectures. Systems with extensive maintainability requirements may not be well suited to client-based or thick client-server architectures because of the need to reinstall software on the desktops.

Performance Requirements Generally speaking, information systems that have high performance requirements are best suited to client-server architectures. Client-server architectures are more scalable, which mean they respond better to changing capacity needs and thus enable the organization to better tune the hardware to the speed requirements of the system. Client-server architectures that have multiple servers in each tier should be more reliable and have greater availability, because if any one server crashes, requests are simply passed to other servers, and users might not even notice (although response time could be worse). In practice, however, reliability and availability depend greatly on the hardware and operating system, and Windows-based computers tend to have lower reliability and availability than Linux or mainframe computers.

Security Requirements Generally speaking, server-based architectures tend to be more secure because all software is in one location and because mainframe operating systems are more secure than microcomputer operating systems. For this reason, high-value systems are more likely to be found on mainframe computers, even if the mainframe is used as a

Requirements	Server-Based	Client-Based	Thin Client-Server	Thick Client-Server
Operational Requirements				
System Integration Requirements	✓		✓	✓
Portability Requirements			✓	
Maintainability Requirements	✓		✓	
Performance Requirements				
Speed Requirements			✓	✓
Capacity Requirements			✓	✓
Availability/Reliability Requirements	✓		✓	✓
Security Requirements				
High System Value	✓		✓	
Access Control Requirements	✓			
Encryption/Authentication Requirements			✓	✓
Virus Control Requirements	✓			
Cultural/Political Requirements				
Multilingual Requirements			✓	
Customization Requirements			✓	
Making Unstated Norms Explicit			✓	
Legal Requirements	✓		✓	✓

FIGURE 13-16
Nonfunctional Requirements and Their Implications for Architecture Design

server in client–server architectures. In today’s Internet-dominated world, authentication and encryption tools for Internet-based client–server architectures are more advanced than those for mainframe-based server-based architectures. Viruses are potential problems in all architectures because they easily spread on desktop computers. If server-based system can reduce the functions needed on desktop systems then they may be more secure.

Cultural and Political Requirements As the cultural and political requirements become more important, the ability to separate the presentation logic from the application logic and the data becomes important. Such separation makes it easier to develop the presentation logic in different languages while keeping the application logic and data the same. It also makes it easier to customize the presentation logic for different users and to change it to better meet cultural norms. To the extent that the presentation logic provides access to the application and data, it also makes it easier to implement different versions that enable or disable different features required by laws and regulations in different countries. This separation is the easiest in thin client–server architectures, so systems with many cultural and political requirements often use thin client–server architectures. As with system integration requirements, the impact of legal requirements depends on the specific nature of the requirements, but in general, client-based systems tend to be less flexible.

**YOUR
TURN**

13-3 University Course Registration System

Think about the course registration system in your university. First, develop a set of nonfunctional requirements if the system were to be developed today. Consider the operational

requirements, performance requirements, security requirements, and cultural and political requirements. Then create an architecture design to satisfy these requirements.

**YOUR
TURN**

13-4 Global e-Learning System

Many multinational organizations provide global Web-based e-learning courses to their employees. First, develop a set of nonfunctional requirements for such a system. Consider the operational requirements,

performance requirements, security requirements, and cultural and political requirements. Then create an architecture design to satisfy these requirements.

HARDWARE AND SOFTWARE SPECIFICATION

The time to begin acquiring the hardware and software that will be needed for the future system is during the design of the system. In many cases, the new system will simply run on the existing equipment in the organization. Other times, however, new hardware and software (usually for servers) must be purchased. The *hardware and software specification* is a document that describes what hardware and software are needed to support the application. The actual acquisition of hardware and software should be left to the purchasing department or the area in the organization that handles capital procurement. However, the project team can use the hardware and software specification to communicate the project needs to the appropriate people. There are several steps involved in creating the document. Figure 13-17 shows a sample hardware and software specification.

First, you will need to define the software that will run on each component. This usually starts with the operating system (e.g., Windows, Linux) and includes any special purpose software on the client and servers (e.g., Oracle database). This should consider any additional costs, such as technical training, maintenance, extended warranties, and licensing agreements (e.g., a site license for a software package). Again, the needs that you list are influenced by decisions that are made in the other design-phase activities.

Second, you must create a list of the hardware that is needed to support the future system. The low-level network model provides a good starting point for recording the project's hardware needs because each of the components on the diagram corresponds to an item on this list. In general, the list can include things like database servers, network servers, peripheral devices (e.g., printers, scanners), backup devices, storage components, and any other hardware component that is needed to support an application. At this time, you also should note the quantity of each item that will be needed.

Third, you must describe, in as much detail as possible, the minimum requirements for each piece of hardware. Typically, the project team must convey requirements like the amount of processing capacity, the amount of storage space, and any special features that should be included. Many organizations have standard lists of approved hardware and software that must be used, so in many cases, this step simply involves selecting items from the lists. Other times, however, the team is operating in new territory and not constrained by

	Standard Client	Standard Web Server	Standard Application Server	Standard Database Server
Operating System	<ul style="list-style-type: none"> Windows Netscape 	<ul style="list-style-type: none"> Linux 	<ul style="list-style-type: none"> Linux 	<ul style="list-style-type: none"> Linux
Special Software	<ul style="list-style-type: none"> Adobe Acrobat Reader Real Audio 	<ul style="list-style-type: none"> Apache 	<ul style="list-style-type: none"> Java 	<ul style="list-style-type: none"> Oracle
Hardware	<ul style="list-style-type: none"> 40 gig disk drive Pentium 17 inch Monitor 	<ul style="list-style-type: none"> 80 gig disk drive Pentium 	<ul style="list-style-type: none"> 80 gig disk drive Pentium 	<ul style="list-style-type: none"> 200 gig disk drive RAID Quad Pentium
Network	<ul style="list-style-type: none"> Always-on Broadband preferred Dial-up at 56Kbps possible with some performance loss 	<ul style="list-style-type: none"> Dual 100 Mbps Ethernet 	<ul style="list-style-type: none"> Dual 100 Mbps Ethernet 	<ul style="list-style-type: none"> Dual 100 Mbps Ethernet

FIGURE 13-17 Sample Hardware and Software Specification

the need to select from an approved list. In these cases, the project team must convey such requirements as the amount of processing capacity, the amount of storage space, and any special features that should be included.

This step becomes easier with experience; however, there are some hints that can help you describe hardware needs (see Figure 13-18). For example, consider the hardware standards within the organization or those recommended by vendors. Talk with experienced system developers or other companies with similar systems. Finally, think about the factors that affect hardware performance, such as the response time expectations of the users, data volumes, software memory requirements, the number of users accessing the system, the number of external connections, and growth projections.

APPLYING THE CONCEPTS AT CD SELECTIONS

Alec Adams, senior systems analyst and project manager for CD Selections' Internet sales system, realized that the hardware, software, and networks that would support the new application would need to be integrated into the current infrastructure at CD Selections.

- 1. Functions and Features** What specific functions and features are needed (e.g., size of monitor, software features)
- 2. Performance** How fast the hardware and software operates (e.g., processor, number of database writes per second)
- 3. Legacy Databases and Systems** How well the hardware and software interacts with legacy systems (e.g., can it write to this database)
- 4. Hardware and OS Strategy** What are the future migration plans (e.g., the goal is to have all of one vendor's equipment)
- 5. Cost of Ownership** What are the costs beyond purchase (e.g., incremental license costs, annual maintenance, training costs, salary costs)
- 6. Political Preferences** People are creatures of habit and are resistant to change, so changes should be minimized
- 7. Vendor Performance** Some vendors have reputations or future prospects that are different from those of a specific hardware or software system they currently sell

FIGURE 13-18
Factors in Hardware and Software Selection

**YOUR
TURN****13-5 University Course Registration System**

Develop a hardware and software specification for the university course registration system described in Your Turn 13-3.

**YOUR
TURN****13-6 Global e-Learning System**

Develop a hardware and software specification for the global e-learning system described in Your Turn 13-4.

**YOUR
TURN****13-7 Create a Hardware and Software Specification**

You have decided to purchase a computer, printer, and low-cost scanner to support your academic work. Create

a hardware and software specification for these components that describes your hardware and software needs.

He began by reviewing the high-level nonfunctional requirements developed in the analysis phase (see Figure 5-13 in Chapter 5) and by conducting a JAD session and a series of interviews with managers in the marketing department and three store managers to refine the nonfunctional requirements into more detail. Figure 13-19 shows some of the results. The clear business need for a Web-based architecture required a thin client-server architecture for the Internet sales portion of the system.

CD Selections had a formal architecture group responsible for managing CD Selections architecture and its hardware and software infrastructure. Therefore, Alec set up a meeting with the project team and the architecture group. During the meeting, he confirmed that CD Selections was still moving toward a target client-server architecture, although the central mainframe still existed as the primary server for many server-based applications.

They discussed the Internet sales system and decided that it should be built using a three-tier thin client-server architecture. Everyone believed that it was hard to know at this point exactly how much traffic this Web site would get and how much power the system would require, but a client-server architecture would allow CD Selections to easily scale up the system as needed.

By the end of the meeting, it was agreed that a three-tiered client-server architecture was the best configuration for the Internet portion of the Internet sales system (i.e., the Place Order process in Figures 6-15 and 6-17). Customers would use their personal computers running a Web browser as the client. A database server would store the Internet system's databases; whereas, an application server would have Web server software and the application software to run the system.

A separate two-tier client server system will maintain the CD and Marketing Material information (i.e., the Maintain CD Information and Maintain Marketing Information processes in Figure 6-17). This system will have an application for the personal computers of the staff working in the Internet sales group that communicates directly with the

1. Operational Requirements

- | | |
|-----------------------|--|
| Technical Environment | <ul style="list-style-type: none"> 1.1 The system will work over the Web environment with Netscape and real audio. 1.2 Customers will only need Netscape and RA on their desktops. |
| System Integration | <ul style="list-style-type: none"> 1.3 The Internet sales system will read information from the main CD information database, which contains basic information about the CD (e.g., title, artist, id number, price, quantity in inventory). The Internet order system will not write information to the main CD information database. 1.4 The Internet sales system will transmit orders for new CDs in the special order system, and will rely on the special order system to complete the special orders generated. 1.5 The Internet sales system will read and write to the main inventory database. 1.6 A new module for the In-store system will be written to manage the “holds” generated by the Internet system. The requirements for this new module will be documented as part of the Internet sales system because they are necessary for the Internet sales system to function. 1.7 A new module will be written to handle the mail order sales. The requirements for this new module will be documented as part of the Internet sales system because they are necessary for the Internet sales system to function. |
| Portability | <ul style="list-style-type: none"> 1.8 The system will need to remain current with evolving Web standards, especially those pertaining to music formats. |
| Maintainability | <ul style="list-style-type: none"> 1.9 No special maintainability requirements are anticipated. |

2. Performance Requirements

- | | |
|------------------------------|---|
| Speed | <ul style="list-style-type: none"> 2.1 Response times must be less than 7 seconds. 2.2 The inventory database must be updated in real time. 2.3 In-store holds must be sent to the store within 5 minutes. |
| Capacity | <ul style="list-style-type: none"> 2.4 There will be a maximum of 20–50 simultaneous users at peak use times. 2.5 The system will support streaming audio to up to forty simultaneous users. 2.6 The system will send up to 5K of data to each store daily. 2.7 The in-store hold database will require 10–20K of disk space per store. |
| Availability and Reliability | <ul style="list-style-type: none"> 2.8 The system should be available 24/7. 2.9 The system shall have 99 percent uptime performance. |

3. Security Requirements

- | | |
|---------------------------|--|
| System Value | <ul style="list-style-type: none"> 3.1 No special system value requirements are anticipated. |
| Access Control | <ul style="list-style-type: none"> 3.2 Only store managers will be able to override In-Store Holds. |
| Encryption/Authentication | <ul style="list-style-type: none"> 3.3 No special encryption/authentication requirements are anticipated. |
| Virus Control | <ul style="list-style-type: none"> 3.4 No special virus control requirements are anticipated. |

4. Cultural and Political Requirements

- | | |
|----------------|---|
| Multilingual | <ul style="list-style-type: none"> 4.1 No special multilingual requirements are anticipated. |
| Customization | <ul style="list-style-type: none"> 4.2 No special customization requirements are anticipated. |
| Unstated Norms | <ul style="list-style-type: none"> 4.3 No special unstated norms requirements are anticipated. |
| Legal | <ul style="list-style-type: none"> 4.4 No special legal requirements are anticipated. |

FIGURE 13-19 Selected Nonfunctional Requirements for the CD Selections Internet Sales System

database server and enables staff to update the information. The database server will have a separate program to enable it to exchange data with CD Selections' distribution system on the company mainframe. Furthermore, the in-store system was currently built using a two-tier client-server architecture, so the portion of the system responsible for the in-store holds would conform to that architecture.

Next, Alec created a network model to show the major components of the Internet sales system (see Figure 13-20). The Internet sales system is on a separate network segment separated from the CD Selections' main network by a firewall that separates the network from the Internet while granting access to the Web and database servers. The Internet sales system has two parts. A firewall is used to connect the Web/Application server to the Internet, while another firewall further protects the Internet sales group's client computers and database server from the Internet. In order to improve response time, a direct connection is made from the Web/Application server to the database server because these will exchange a lot of data. Based on these decisions, Alec also created a deployment diagram that shows how the problem domain, human computer interaction, and data management layers would be deployed over the physical architecture layer (see Figure 13-21).

Given that the Web interface could reach a geographically dispersed group, the project team realized that it needed to plan for 24/7 system support. He scheduled a meeting to talk with the CD Selections systems operations group and discussed how they might be able to support the Internet sales system outside of standard working hours.

After examining the network model, the architecture group and the Internet project team decided that the only components that needed to be acquired for the project are a database server, a Web server, and five new client computers for the marketing group, who will maintain the marketing materials. They developed a hardware and software specification for these components and handed them off to the purchasing department to start the acquisition process.

SUMMARY

An important component of design is the design of the physical architecture layer, which includes the hardware, software, and communications infrastructure for the new system and the way in which the information system will be distributed over the architecture. The physical architecture layer design is described in a deliverable, which contains the network model and the hardware and software specification.

Elements of the Physical Architecture Layer

All software systems can be divided into four basic functions: data storage, data access logic, application logic, and the presentation logic. There are three fundamental computing architectures that place these functions on different computers. In server-based architectures, the server performs all the functions. In client-based architectures, the client computers are responsible for presentation logic, application logic, and data access logic, with data stored on a file server. In client-server architectures, the client is responsible for the presentation logic and the server is responsible for the data access logic and data storage. In thin client-server architectures, the server performs the application logic, while in thick client-server architectures, the application logic is shared between the servers and clients. In a two-tiered client-server architecture there are two groups of computers: one client and a set of servers. In a three-tiered client-server architecture, there are three groups of computers: a client, a set of application servers, and a set of database servers. From an object-oriented point of view, the future trend in client-server computing is distributed object computing (DOC). Currently, there are three competing approaches to support DOC: OMG's CORBA, Sun's EJB and J2EE, and Microsoft's DCOM and .net.

Each of the computing architectures has its strengths and weaknesses, and no architecture is inherently better than the others. The choice that the project team makes should be based on several criteria, including cost of development, ease of development, need for GUI applications, network capacity, central control and security, and scalability. The project team should also take into consideration the existing architecture and special software requirements of the project.

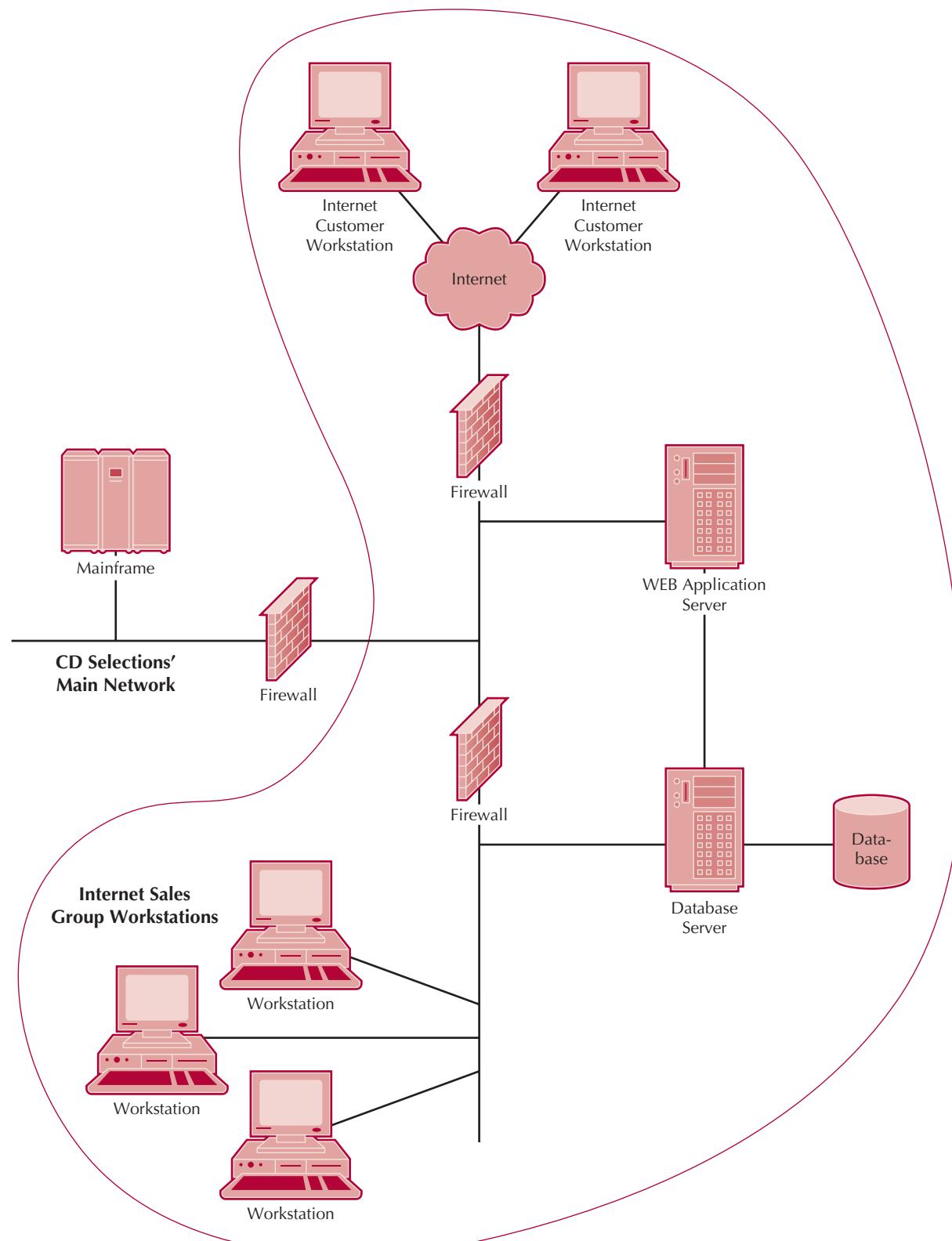


FIGURE 13-20 Deployment Diagram of Network Model for the CD Selections Internet Sales System

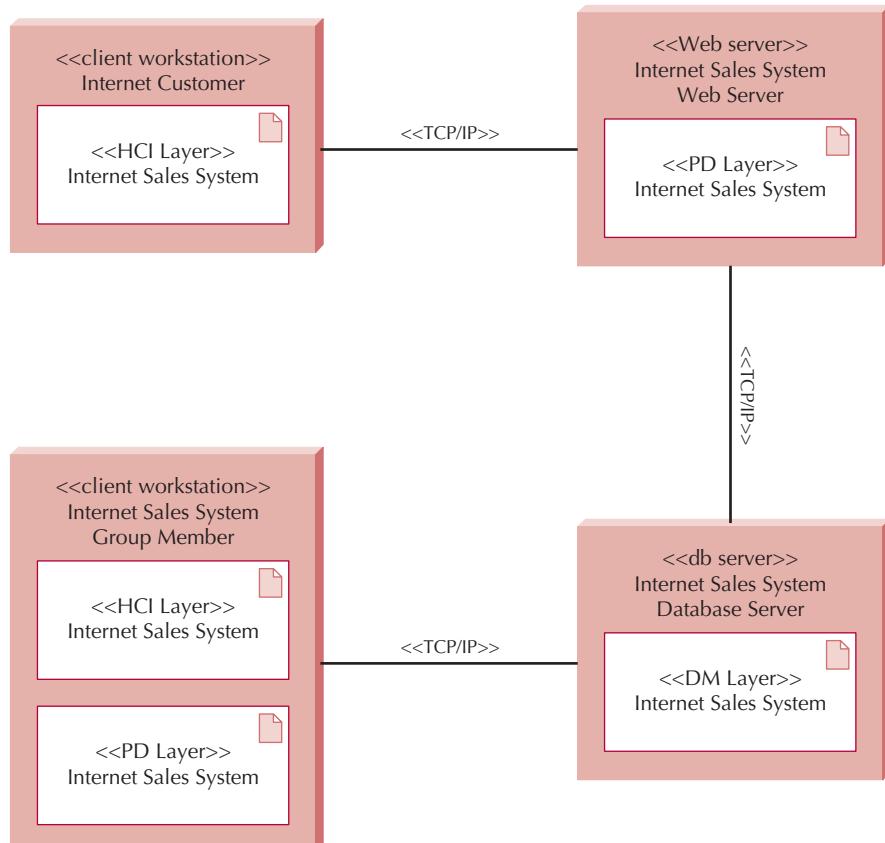


FIGURE 13-21
Deployment Diagram
of Layers for the CD
Selections Internet
Sales System

Infrastructure Design

Deployment diagrams are used to portray the design of the physical architecture layer. The diagrams are composed of nodes, artifacts, and communication links. Also, the diagram can be extended with graphical icons that represent different types of nodes.

The network model is a diagram that shows the technical components of the information system (e.g., servers, personal computers, networks) and their geographic locations throughout the organization. The components of the network model are the various clients (e.g., personal computers, kiosks), servers (e.g., database, network, communications, printer), network equipment (e.g., wire, dial-in connections, satellite links), and external systems or networks (e.g., Internet service provider) that support the application. Creating a network model is a top-down process whereby a high-level diagram is first created to show the geographic sites, or locations, that house the various components of the future system. Next, a low-level diagram is created to describe each location in detail, and it shows the system's hardware components and how they are attached to one another.

Nonfunctional Requirements and Physical Architecture Layer Design

Creating an architecture design begins with the nonfunctional requirements. Operational requirements specify the operating environment(s) in which the system must perform and how those may change over time (i.e., technical environment, system integration, portability, and maintainability). Performance requirements focus on performance issues such as system speed, capacity, and availability and reliability. Security requirements attempt to protect the information system from disruption and data loss (e.g., system value, access

control, encryption and authentication, and virus control). Cultural and political requirements are those that are specific to the specific countries in which the system will be used (e.g., multilingual, customization, unstated norms, and legal).

Hardware and Software Specification

The hardware and software specification is a document that describes what hardware and software are needed to support the application. To create a specification document, the hardware that is needed to support the future system is listed and then described in as much detail as possible. Next, the software to run on each hardware component is written down, along with any additional associated costs, such as technical training, maintenance, extended warranties, and licensing agreements. Although the project team may suggest specific products or vendors, ultimately the hardware and software specification is turned over to the people who are in charge of procurement.

KEY TERMS

.net	Distributed objects computing (DOC)	Performance requirements
24/7		Physical architecture layer design
Access control requirements	Encryption	Portability requirements
Application logic	Encryption and authentication requirements	Presentation logic
Architectural component	Enterprise JavaBeans (EJB)	Private key
Artifact	Fat client	Public key
Asymmetric encryption algorithm	Graphical user interface (GUI)	Public key encryption
Authentication	Hardware and software specification	Response time
Availability and reliability requirements	Invertible	Scalable
Capacity requirements	Java 2 Enterprise Edition (J2EE)	Security requirements
Certificate authority (CA)	Legal requirements	Server
Client computer	Locations	Server-based architecture
Client-based architecture	Mainframe	Speed requirements
Client-server architecture	Maintainability requirements	Structured query language (SQL)
Common Object Request Broker Architecture (CORBA)	Microcomputer	Symmetric encryption algorithm
Communication path	Middleware	System integration requirements
Cultural and political requirements	Minicomputer	Technical environment requirements
Customization requirements	Mission critical system	Thick client
Data access logic	Multilingual requirements	Thin client
Data storage	Network	Three-tiered architecture
Deployment diagrams	Network model	Total cost of ownership
Distributed Computing Object Model (DCOM)	N-tiered architecture	Two-tiered architecture
	Node	Unstated norms
	Operational requirements	Virus
		Virus control requirements

QUESTIONS

- What are the four basic functions of any information system?
- What are the three primary hardware components of any physical architecture?
- Name two examples of a server.
- Compare and contrast sever-based architectures, client-based architectures, and client-sever-based architectures.
- What is the biggest problem with server-based computing?
- What is the biggest problem with client-based computing?

7. Describe the major benefits and limitations of thin client-server architectures.
8. Describe the major benefits and limitations of thick client-server architectures.
9. Describe the differences among two-tiered, three-tiered, and n-tiered architectures.
10. What is distributed object computing?
11. What are the three competing approaches to support distributed object computing?
12. Define scalable. Why is this term important to system developers?
13. What six criteria are helpful to use when comparing the appropriateness of computing alternatives?
14. Why should the project team consider the existing physical architecture in the organization when designing the physical architecture layer of the new system?
15. Describe the major nonfunctional requirements and how they influence physical architecture layer design.
16. Why is it useful to define the nonfunctional requirements in more detail even if the technical environment requirements dictate a specific architecture?
17. What does the network model communicate to the project team?
18. What are the differences between the top-level network model and the low-level network model?
19. What additional hardware- and software-associated costs may need to be included on the hardware and software specification?
20. Who ultimately is in charge of acquiring hardware and software for the project?
21. What do you think are three common mistakes that novice analysts make in physical architecture design and hardware and software specification?
22. Are some nonfunctional requirements more important than others in influencing the physical architecture design and hardware and software specification?
23. What do you think are the most important security issues for a system?

EXERCISES

- A. Using the Web (or past issues of computer industry magazines such as *Computerworld*), locate a system that runs in a server-based environment. Based on your reading, why do you think the company chose that computing environment?
- B. Using the Web (or past issues of computer industry magazines such as *Computerworld*), locate a system that runs in a client-server environment. Based on your reading, why do you think the company chose that computing environment?
- C. Using the Web, locate examples of a mainframe component, a minicomputer component, and a microcomputer component. Compare the components in terms of price, speed, available memory, and disk storage. Did you find large differences in prices when the performances of the components are considered?
- D. You have been selected to find the best client-server computing architecture for a Web-based order entry system that is being developed for L.L.Bean. Write a short memo that describes to the project manager your reason for selecting an n-tiered architecture over a two-tiered architecture. In the memo, give some idea as to what the different components of the architecture you would include.
- E. You have been selected to determine whether your firm should invest in CORBA-, EJB/J2EE, or DCOM/.net-based technologies to support distributed objects computing. Write a short memo that describes to the steering committee the advantages and disadvantages of both approaches. (*Note:* you probably should check out the Web sites associated with each.)
- F. Think about the system that your university currently uses for career services, and pretend that you are in charge of replacing the system with a new one. Describe how you would decide on the computing architecture for the new system using the criteria presented in this chapter. What information will you need to find out before you can make an educated comparison of the alternatives?
- G. Locate a consumer products company on the Web and read its company description (so that you get a good understanding of the geographic locations of the company). Pretend that the company is about to create a new application to support retail sales over the Web. Create a high-level network model that depicts the locations that would include components that support this application.
- H. An energy company with headquarters in Dallas, Texas, is thinking about developing a system to track the efficiency of its oil refineries in North America. Each week, the ten refineries as far as Valdez, Alaska, and as close as San Antonio, Texas, will upload performance data via satellite to the corporate mainframe in Dallas. Production managers at each site will use a personal computer to dial into an Internet service

provider and access reports via the Web. Create a high-level network model that depicts the locations that have components supporting this system.

- I. Create a low-level network diagram for the building that houses the computer labs at your university. Choose an application (e.g., course registration, student admissions) and only include the components that are relevant to that application.
- J. Pretend that your mother is a real estate agent, and she has decided to automate her daily tasks using a laptop computer. Consider her potential hardware

and software needs, and create a hardware and software specification that describes them. The specification should be developed to help your mother buy her hardware and software on her own.

- K. Pretend that the admissions office in your university has a Web-based application so that students can apply for admission online. Recently, there has been a push to admit more international students into the university. What do you recommend that the application includes to ensure that it supports this global requirement?

MINICASES

1. The system development project team at Birdie Masters golf schools has been working on defining the physical architecture design for the system. The major focus of the project is a networked school location operations system, allowing each school location to easily record and retrieve all school location transaction data. Another system element is the use of the Internet to enable current and prospective students to view class offerings at any of the Birdie Masters' locations, schedule lessons and enroll in classes at any Birdie Masters location, and maintain a student progress profile—a confidential analysis of the student's golf skill development.

The project team has been considering the globalization issues that should be factored into the architecture design. The school's plan for expansion into the golf-crazed Japanese market is moving ahead. The first Japanese school location is tentatively planning to open about six months after the target completion date for the system project. Therefore, it is important that issues related to the international location be addressed now during design.

Assume that you have been given the responsibility of preparing a summary memo on the globalization issues that should be factored into the design. Prepare

this memo discussing the globalization issues that are relevant to Birdie Masters' new system.

2. Jerry is a relatively new member of a project team that is developing a retail store management system for a chain of sporting goods stores. Company headquarters is in Las Vegas, and the chain has twenty-seven locations throughout Nevada, Utah, and Arizona. Several cities have multiple stores.

The new system will be a networked, client-server architecture. Stores will be linked to one of three regional servers, and the regional servers will be linked to corporate headquarters in Las Vegas. The regional servers also link to each other. Each retail store will be outfitted with similar configurations of two PC-based point-of-sale terminals networked to a local file server. Jerry has been given the task of developing a network model that will document the geographic structure of this system. He has not faced a system of this scope before and is a little unsure how to begin.

- a. Prepare a set of instructions for Jerry to follow in developing this network model.
- b. Using a deployment diagram, draw a network model for this organization.
- c. Prepare a set of instructions for Jerry to follow in developing a hardware and software specification.

PART FOUR

IMPLEMENTATION PHASE

During the implementation phase, the actual system is built. Building a successful information system requires a set of activities: programming, testing, and documenting the system. Installing an information system requires switching from the current system to the new system. This conversion process can be quite involved. Not only does it involve shutting the old system down and turning the new one on, it also can involve a significant training effort. Finally, operating the system may uncover additional requirements that can be fed back into the next build.

CHAPTER 14

CONSTRUCTION

This chapter discusses the activities needed to successfully build the information system: programming, testing, and documenting the system. Programming is time-consuming and costly, but except in unusual circumstances, it is the simplest for the systems analyst because it is well understood. For this reason, the system analyst focuses on testing (proving that the system works as designed) and developing documentation during this part of the SDLC.

OBJECTIVES

- Be familiar with the system construction process.
- Understand different types of tests and when to use them.
- Understand how to develop documentation.

CHAPTER OUTLINE

Introduction	Acceptance Tests
Managing Programming	Developing Documentation
Assigning Programmers	Types of Documentation
Coordinating Activities	Designing Documentation Structure
Managing the Schedule	Writing Documentation Topics
Designing Tests	Identifying Navigation Terms
Testing and Object-Orientation	Applying the Concepts at CD Selections
Test Planning	Managing Programming
Unit Tests	Testing
Integration Tests	Developing User Documentation
System Tests	Summary

INTRODUCTION

When people first learn about developing information systems, usually they immediately think about writing programs. Programming can be the largest single component of any systems development project in terms of both time and cost. However, it also can be the best understood component and therefore—except in rare circumstances—offers the least problems of all aspects of the SDLC. When projects fail, it is usually not because the programmers were unable to write the programs but because the analysis, design, installation, and/or project management were done poorly. In this chapter, we focus on the construction and testing of the software and the documentation.

Construction is the development of all parts of the system, including the software itself, documentation, and new operating procedures. Programming is often seen as the focal point of system development. After all, system development *is* writing programs. It is the reason why we do all the analysis and design. And it's fun. Many beginning programmers see testing and documentation as bothersome afterthoughts. Testing and documentation aren't fun, so they often receive less attention than the creative activity of writing programs.

Programming and testing, however, are very similar to writing and editing. No professional writer (or student writing an important term paper) would stop after writing the first draft. Rereading, editing, and revising the initial draft into a good paper is the hallmark of good writing. Likewise, thorough testing is the hallmark of professional software developers. Most professional organizations devote more time and money to testing (and the subsequent revision and retesting) than to writing the programs in the first place.

The reasons are simple economics: downtime and failures caused by software bugs¹ are extremely expensive. Many large organizations estimate the costs of downtime of critical applications at \$50,000 to \$200,000 *per hour*.² One serious bug that causes an hour of downtime can cost more than one year's salary of a programmer—and how often are bugs found and fixed in one hour? Testing is therefore a form of insurance. Organizations are willing to spend a lot of time and money to prevent the possibility of major failures after the system is installed.

Therefore, programs are not usually considered finished until the test for that program is passed. For this reason, programming and testing are tightly coupled, and since programming is the primary job of the programmer (not the analyst), testing (not programming) often becomes the focus of the construction stage for the systems analysis team.

In this chapter, we discuss three parts of the construction step: programming, testing, and writing the documentation. Because programming is primarily the job of programmers, not systems analysts, and because this is not a programming book, we devote less time to programming than to testing and documentation.

MANAGING PROGRAMMING

In general, systems analysts do not write programs; programmers write programs. Therefore, the primary task of the systems analysts during programming is ... waiting. However, the project manager is usually very busy *managing* the programming effort by assigning the programmers, coordinating the activities, and managing the programming schedule.³

Assigning Programmers

The first step in programming is assigning modules to the programmers. As discussed in Chapter 10, each module (class, object, or method) should be as separate and distinct as possible from the other modules. The project manager first groups together classes that are related so that each programmer is working on related classes. These groups of classes are then assigned to programmers. A good place to start is to look at the package diagrams.

¹ When I was an undergraduate, I had the opportunity to hear Admiral Grace Hopper tell how the term bug was introduced. She was working on one of the early Navy computers when suddenly it failed. The computer would not restart properly so she began to search for failed vacuum tubes. She found a moth inside one tube and recorded in the log book that a bug had caused the computer to crash. From then on, every computer crash was jokingly blamed on a bug (as opposed to programmer error), and eventually the term bug entered the general language of computing.

² See Billie Shea, "Quality Patrol: Eye on the Enterprise," *Application Development Trends* (November 5, 1998): 31–38.

³ One of the best books on managing programming (even though it was first written 25 years ago) is that by Frederick P. Brooks, Jr., *The Mythical Man-Month*, 20th Anniversary edition (Reading, MA: Addison-Wesley, 1995).

CONCEPTS**IN ACTION****14-A The Cost of a Bug**

My first programming job in 1977 was to convert a set of application systems from one version of COBOL to another version of COBOL for the government of Prince Edward Island. The testing approach was to first run a set of test data through the old system and then run it through the new system to ensure that the results from the two matched. If they matched, then the last three months of production data were run through both to ensure that they too matched.

Things went well until I began to convert the Gas Tax system that kept records on everyone authorized to purchase gasoline without paying tax. The test data ran fine, but the results using the production data were peculiar. The old and new systems matched, but rather than listing several thousand records, the report listed only fifty. I checked the production data file and found it listed only fifty records, not the thousands that were supposed to be there.

The system worked by copying the existing gas tax records file into a new file and making changes in the new file. The old file was then copied to tape backup. There was a bug in the program such that if there were no changes to the file, a new file was created, but no records were copied into it.

I checked the tape backups and found one with the full set of data that was scheduled to be overwritten three days after I discovered the problem. The government was only three days away from losing all gas tax records.

—Alan Dennis

Question

1. What would have been the cost of this bug if it hadn't been caught?

One rule of system development is that the more programmers who are involved in a project, the longer the system will take to build. This is because as the size of the programming team increases, the need for coordination increases exponentially, and the more coordination that is required, the less time programmers can spend actually writing systems. The best size is the smallest possible programming team. When projects are so complex that they require a large team, the best strategy is to try to break the project into a series of smaller parts that can function as independently as possible.

Coordinating Activities

Coordination can be done through both high-tech and low-tech means. The simplest approach is to have a weekly project meeting to discuss any changes to the system that have arisen during the past week—or just any issues that have come up. Regular meetings, even if they are brief, encourage the widespread communication and discussion of issues before they become problems.

Another important way to improve coordination is to create and follow standards that can range from formal rules for naming files to forms that must be completed when goals are reached to programming guidelines (see Chapter 4). When a team forms standards and then follows them, the project can be completed faster because task coordination is less complex.

The analysts also must put mechanisms in place to keep the programming effort well-organized. Many project teams set up three “areas” in which programmers can work: a development area, a testing area, and a production area. These areas can be different directories on a server hard disk, different servers, or different physical locations, but the point is that files, data, and programs are separated based on their status of completion. At first, programmers access and build files within the development area and then copy them to the testing area when the programmers are “finished.” If a program does not pass a test, it is sent back to development. Once all programs and the like are

tested and ready to support the new system, they are copied into the production area—the location where the final system will reside.

One of the more important things to coordinate is the traceability of the implementation of the system back to the original requirements. Each class and method must be associated with an individual requirement or a set of requirements. In most object-oriented system development approaches today capture and document the requirements in use cases. As such, it is critical to guarantee that all classes and methods can be traced back to an individual use case or a set of use cases. This is why in our approach (MOOSAD), we have included a section for “Associated Use Cases” on the CRC cards (see Figures 7-1 and 10-11) and the contracts (see Figure 10-13).

Keeping files and programs in different places based on completion status helps manage *change control*, the action of coordinating a system as it changes through construction. Another change control technique is keeping track of which programmer changes which classes and packages by using a *program log*. The log is merely a form on which programmers “sign-out” classes and packages to write and “sign-in” when they are completed. Both the programming areas and program log help the analysts understand exactly who has worked on what and to determine the system’s current status. Without these techniques, files can be put into production without the proper testing; two programmers, for example, could start working on the same class or package at the same time.

If a CASE tool is used during the construction step, it can be very helpful for change control because many CASE tools are set up to track the status of programs and help manage programmers as they work. In most cases, maintaining coordination is not conceptually complex. It just requires a lot of attention and discipline to track small details.

Managing the Schedule

The time estimates that were produced during the initial planning phase and refined during the analysis and design phases will almost always need to be refined as the project progresses during construction because it is virtually impossible to develop an exact assessment of the project’s schedule. As we discussed in Chapter 4, a well-done set of time estimates will usually have a 10 percent margin of error by the time you reach the construction step. It is critical that the time estimates be revised as the construction step proceeds. If a program module takes longer to develop than expected, then the prudent response is to move the expected completion date later by the same amount.

One of the most common causes of schedule problems is scope creep. *Scope creep* occurs when new requirements are added to the project after the system design was finalized. Scope creep can be very expensive because changes made late in the SDLC can require much of the completed system design (and even programs already written) to be redone. Any proposed change during the construction phase must require the approval of the project manager and should only be done after a quick cost-benefit analysis has been done.

Another common cause is the unnoticed day-by-day *slippage* in the schedule. One package is a day later here, another one a day late there. Pretty soon these minor delays add up, and the project is noticeably behind schedule. Once again, the key to managing the programming effort is to watch these minor slippages carefully and update the schedule accordingly.

Typically, a project manager will create a *risk assessment* that tracks potential risks along with an evaluation of their likelihood and potential impact. As the construction step moves to a close, the list of risks will change as some items are removed and others surface. The best project managers, however, work hard to keep risks from having an impact on the schedule and costs associated with the project.

PRACTICAL**TIP**

In previous chapters, we discussed classic mistakes and how to avoid them. Here, we summarize four classic mistakes in the implementation phase.⁴

1. Research-oriented Development: Using state-of-the-art technology requires research-oriented development that explores the new technology because “bleeding edge” tools and techniques are not well understood, are not well documented, and do not function exactly as promised.

Solution: If you use state-of-the-art technology, you need to significantly increase the project’s time and cost estimates even if (some experts would say *especially if*) such technologies claim to reduce time and effort.

2. Using Low-Cost Personnel: You get what you pay for. The lowest cost consultant or the staff member is significantly less productive than the best staff. Several studies have shown that the best programmers produce software six to eight times faster than the least productive (yet cost only 50 to 100 percent more).



Solution: If cost is a critical issue, assign the best, most expensive personnel; never assign entry-level personnel in an attempt to save costs.

3. Lack of Code Control: On large projects, programmers need to coordinate changes to the program source code (so that two programmers don’t try to change the same program at the same time and overwrite the other’s changes). Although manual procedures appear to work (e.g., sending e-mail notes to others when you work on a program to tell them not to), mistakes are inevitable.

Solution: Use a source code library that requires programmers to “check out” programs and prohibits others from working on them at the same time.

4. Inadequate Testing: The number one reason for project failure during implementation is ad hoc testing—where programmers and analysts test the system without formal test plans.

Solution: Always allocate sufficient time in the project plan for formal testing.

DESIGNING TESTS⁵

In object-oriented systems, the temptation is to minimize testing. After all, through the use of patterns, frameworks, class libraries, and components, much of the system has been tested previously. Therefore, we should not have to test as much. Right? Wrong!! Testing is more critical to object-oriented systems than to systems developed in the past. Based on encapsulation (and information hiding), polymorphism (and dynamic binding), inheritance, and reuse, thorough testing is much more difficult and critical. Testing must be done systematically and the results documented so that the project team knows what has and has not been tested.

The purpose of testing is not to demonstrate that the system is free of errors. Indeed, it is not possible to prove that a system is error free, especially object-oriented systems. This is similar to theory testing. You cannot prove a theory. If a test fails to find problems with a theory, your confidence in the theory is increased. However, if a test succeeds in finding a problem, then the theory has been falsified. Software testing is similar in that it can only show the existence of errors. As such, the purpose of testing is to uncover as many errors as feasible.⁶

⁴ Adapted from Steve McConnell, *Rapid Development* (Redmond, WA: Microsoft Press, 1996).

⁵ This section is based on material contained in Imran Bashir and Amrit L. Goel, *Testing Object-Oriented Software: Life Cycle Solutions* (New York: Springer Verlag, 1999); Bernd Bruegge and Allen H. Dutoit, *Object-Oriented Software Engineering: Conquering Complex and Changing Systems*, (Englewood Cliffs, NJ: Prentice Hall, 2000); Philippe Kruchten, *The Rational Unified Process: An Introduction*, 2nd ed., (Reading, MA: Addison-Wesley, 2000); and John D. McGregor and David A. Sykes, *A Practical Guide to Testing Object-Oriented Software* (Reading, MA: Addison-Wesley, 2001). For more information on testing object-oriented software, see Robert V. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools* (Reading, MA: Addison-Wesley, 1999); and Shel Sieget, *Object-Oriented Software Testing: A Hierarchical Approach* (New York: Wiley, 1996).

⁶ It is not cost-effective to get each and every error out of the software. Except in simple examples, it is in fact impossible. There are simply too many combinations to check.

There are four general stages of tests: unit tests, integration tests, system tests, and acceptance tests. Although each application system is different, most errors are found during integration and system testing (see Figure 14-1).

In each of the following sections, we describe the four stages. However, before doing this, we describe the effect that the object-oriented characteristics have on testing and the necessary planning and management activities that must take place to have a successful testing program.

Testing and Object-Orientation

Most testing techniques have been developed to support nonobject-oriented development. Object-oriented development is still relatively new. As such, most of the testing approaches have had to be adapted to object-oriented systems. The characteristics of object-oriented systems that affect testing the most are encapsulation (and information hiding), polymorphism (and dynamic binding), inheritance, and the use of patterns, class libraries, frameworks, and components. Also, the sheer volume of products that come out of a typical object-oriented development process has increased the importance of testing in object-oriented development.

Encapsulation and Information Hiding Encapsulation and information hiding allow processes and data to be combined to create holistic entities (i.e., objects). Furthermore, they support hiding everything behind a visible interface. Although this allows the system to be modified and maintained in an effective and efficient manner, it makes testing the system problematic. What do you need to test to build confidence in the system's ability to meet the user's need? Answer, you need to test the business process that is represented in the use cases. However, the business process is distributed over a set of collaborating classes and contained in the methods of those classes. The only way to know the effect that a business process has on a system is to look at the state changes that take place in the system. But in object-oriented systems, the instances of the classes hide the data behind a class boundary. How is it possible then to see the impact of a business process?

A second issue raised by encapsulation and information hiding is the definition of a "unit" for unit testing. What is the unit to be tested? Is it the package, class, or method? In

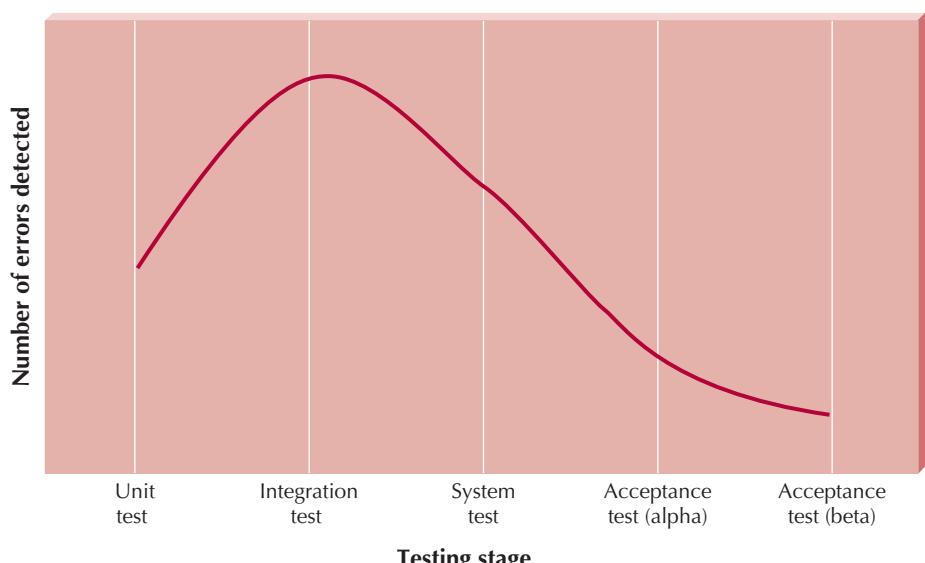


FIGURE 14-1
Error discovery rates for different stages of tests

traditional approaches, the answer would be the process that is contained in a function. However, the process in object-oriented systems is distributed over a set of classes. Therefore, testing individual methods makes no sense. The answer is the class. This dramatically changes the manner in which unit testing is done.

A third issue raised is the impact on integration testing. In this case, objects can be aggregated to form aggregate objects; for example, a car has many parts, or they can be grouped together to form collaborations. Furthermore, they can be used in class libraries, frameworks, and components. Based on all of these different ways in which classes can be grouped together, how does one effectively do integration testing?

Polymorphism and Dynamic Binding Polymorphism and dynamic binding dramatically impact both unit and integration testing. Since an individual business process is implemented through a set of methods distributed over a set of objects, as shown above, the unit test makes no sense to be at the method level. However, with polymorphism and dynamic binding, the same method (a small part of the overall business process) can be implemented in many different objects. Therefore, testing individual implementations of methods makes no sense. Again, the unit that makes sense to test is the class. Furthermore, except for trivial cases, dynamic binding makes it impossible to know which implementation is going to be executed until the system does it. Therefore, integration testing becomes very challenging.

Inheritance When considering the issues raised about inheritance (see Chapter 10), it should be no surprise that inheritance impacts the testing of object-oriented systems. Through the use of inheritance, bugs can be propagated from a superclass to all of its direct and indirect subclasses instantaneously. However, the tests that are applicable to a superclass are also applicable to all of its subclasses. As usual, inheritance is a double-edged sword. Finally, even though we have stated this many times before, inheritance should only support a generalization and specialization type of semantics. Remember, when using inheritance the principle of substitutability is critical (see Chapter 7). All of these issues impact unit and integration testing.

Reuse On the surface, reuse should decrease the amount of testing required. However, each time a class is used in a different context, the class must be tested again. Therefore, anytime a class library, framework, or component is used, unit testing and integration testing are important. In the case of a component, the unit to test is the component itself. Remember that a component has a well-defined Application Program Interface (API) that hides the details of its implementation.

Object-Oriented Development Process and Products In virtually all textbooks, including this one, testing is always covered near the end of the SDLC. This seems to imply that testing takes place only after the programming has ended. However, each and every product⁷ that comes out of the object-oriented development process must be tested. For example, it is a lot easier to ensure that the requirements are captured and modeled correctly through testing the use cases. Furthermore, it is a lot cheaper to catch this type of error back in the analysis phase than it is in the implementation phase. Obviously, this is true for testing collaborations as well. By the time we have implemented a collaboration as a set of layers and partitions, we could have expended a great deal of time, and time is money, on implementing the wrong thing. So, testing collaborations by role-playing the CRC cards back in the analysis phase will actually save the team lots of time and money.

⁷ For example, use-case descriptions, use-case diagrams, CRC cards, class diagrams, object diagrams, sequence diagrams, communication diagrams, behavioral state machines, package diagrams, use scenarios, window navigation diagrams, real use cases, contracts, method specifications, and source code.

Testing must take place throughout the SDLC, not simply at the end. However, the type of testing that can take place on nonexecutable representations, such as use cases and CRC cards, is different from those that take place on code written in an object-oriented programming language. The primary approach to testing nonexecutable representations is some form of an inspection or walkthrough of the representation.⁸ Role-playing the CRC cards based on use cases is an example of one type of walkthrough.

Test Planning

Testing starts with the development of a *test plan* that defines a series of *tests* that will be conducted. Since testing takes place through the development of an object-oriented system, a test plan should be developed at the very beginning of the SDLC and continuously updated as the system evolves. The test plan should address all products that are created during the development of the system. For example, tests should be created that can be used to test the completeness of a CRC card. Figure 14-2 shows a typical unit test plan form for a class. For example, Figure 14-3 shows a partial list of the invariant test specifications for a class. Each individual test has a specific objective and describes a set of very specific *test cases* to examine. In the case of invariant-based tests, a description of the invariant, the original values of the attribute, the event that will cause the attribute value to change, the actual results observed, the expected results, and whether it passed or failed are shown. *Test specifications*

Class Test Plan		Page _____ of _____
Class Name: _____		Version Number: _____ CRC Card ID: _____
Tester: _____		Date Designed: _____ Date Conducted: _____
Class Objective:		
Associated Contract IDs: _____		
Associated Use Case IDs: _____		
Associated Superclass(es): _____		
Testing Objectives:		
Walkthrough Test Requirements:		
Invariant-Based Test Requirements:		
State-Based Test Requirements:		
Contract-Based Test Requirements:		

TEMPLATE
can be found at
[www.wiley.com/
college/dennis](http://www.wiley.com/college/dennis)

FIGURE 14-2
Class Test Plan

⁸ See Michael Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, 15, no. 3 (1976): 105–211.

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 14-3
Class Invariant Test Specification

Class Invariant Test Specification					
Page _____ of _____					
Class Name: _____		Version Number: _____		CRC Card ID: _____	
Tester: _____		Date Designed: _____		Date Conducted: _____	
Testing Objectives:					
Test Cases					
Invariant Description	Original Attribute Value	Event	New Attribute Value	Expected Result	Result P/F
Attribute Name:					
1)	_____	_____	_____	_____	_____
2)	_____	_____	_____	_____	_____
3)	_____	_____	_____	_____	_____
Attribute Name:					
1)	_____	_____	_____	_____	_____
2)	_____	_____	_____	_____	_____
3)	_____	_____	_____	_____	_____
Attribute Name:					
1)	_____	_____	_____	_____	_____
2)	_____	_____	_____	_____	_____
3)	_____	_____	_____	_____	_____

are created for each type of constraint that must be met by the class. Also, similar types of specifications are done for integration, system, and acceptance tests.

Not all classes are likely to be finished at the same time, so the programmer usually writes *stubs* for the unfinished classes to enable the classes around them to be tested. A stub is a placeholder for a class that usually displays a simple test message on the screen or returns some *hardcoded value*⁹ when it is selected. For example, consider an application system that provides the five standard processes discussed in Chapter 6 for some object such as CDs, patients, or employees: creating, changing, deleting, finding, and printing (whether on the screen or on a printer). Depending on the final design, these different processes could end up in different objects on different layers. Therefore, to test the functionality associated with the classes on the problem domain layer, a stub would be written for each of the classes on the other layers that interact with the problem domain classes. These stubs would contain the minimal interface necessary to be able to test the problem domain classes. For example, they would have methods that could receive the messages being sent by the problem domain layer objects and methods that could send messages to the problem domain layer objects. Typically, the methods would display a message on the screen notifying the tester that the method had been successfully reached (e.g., “Delete item from

⁹ The word “hardcoded” means written into the program. For example, suppose you were writing a unit to calculate the net present value of a loan. The stub might be written so that it would always display (or return to the calling module) a value of 100 regardless of the input values.

Database method reached”). In this way, the problem domain classes could pass class testing before the classes on the other layers were completed.

Unit Tests

Unit tests focus on a single unit—the class. There are two approaches to unit testing: black-box and white-box (see Figure 14-4). *Black-box testing* is the most commonly used since each class represents an encapsulated object. Black-box testing is driven by the CRC cards, behavioral state machines, and contracts associated with a class, not by the programmers’ interpretation. In this case, the test plan is developed directly from the specification of the class: each item in the specification becomes a test, and several test cases are developed for it. *White-box testing* is based on the method specifications associated with each class. However, white-box testing has had limited impact in object-oriented development. This is due to the rather small size of the individual methods in a class. As such, most approaches to testing classes use black-box testing to assure their correctness.

Class tests should be based on the invariants on the CRC cards, the behavioral state machines associated with each class, and the pre- and post-conditions contained on each method’s contract. Assuming all of the constraints have been captured on the CRC cards and contracts, individual test cases can be developed fairly easily. For example, suppose the CRC card for an order class gave an invariant that the order quantity must be between 10 and 100 cases. The tester develops a series of test cases to ensure that the quantity is validated before the system accepts it. It is impossible to test every possible combination of input and situation; there are simply too many possible combinations. In this example, the test requires a minimum of three test cases: one with a valid value (e.g., 15), one with an invalid value too low (e.g., 7), and one with invalid value too high (e.g., 110). Most tests would also include a test case with a nonnumeric value to ensure the data types were checked (e.g., ABCD). A really good test would include a test case with nonsensical but potentially valid data (e.g., 21.4).

Using a behavioral state machine is a useful way to identify tests for a class. Any class that has a behavioral state machine associated with it will have a potentially complex life cycle. As such, it is possible to create a series of tests to guarantee that each state can be reached.

Tests also can be developed for each contract associated with the class. In the case of a contract, a set of tests for each pre- and post-condition is required. Furthermore, if the class is a subclass of another class, then all of the tests associated with the superclass must be executed again. Also, the interactions among the constraints, invariants, and pre- and post-conditions in the subclass and the superclass(es) must be addressed.

Finally, owing to good object-oriented design, in order to fully test a class, special testing methods may have to be added to the class being tested. For example, how can invariants be tested? The only way to really test them is through methods that are visible to the outside of the class that can be used to manipulate the values of the class’s attributes. However, adding these types of methods to a class does two things. First, they add to the testing requirements since they themselves will have to be tested. Second, if they are not removed from the deployed version of the system, the system will be less efficient and the advantage of information hiding effectively will be lost. As is readily apparent, testing classes is complex. Therefore, great care must be taken when designing tests for classes.

Integration Tests

Integration tests assess whether a set of classes that must work together do so without error. They ensure that the interfaces and linkages between different parts of the system work properly. At this point, the classes have passed their individual unit tests, so the focus now is on the flow of control among the classes and on the data exchanged among them. Integration testing follows the same general procedures as unit testing: the tester develops a

Stage	Types of Tests	Test Plan Source	When to Use	Notes
Unit Testing	Black-Box Testing Treats class as "black-box"	CRC Cards Class Diagrams Contracts	For normal unit testing	<ul style="list-style-type: none"> Tester focuses on whether the class meets the requirements stated in the specifications.
	White-Box Testing Looks inside the class to test its major elements	Method Specifications	When complexity is high	<ul style="list-style-type: none"> By looking inside the class to review the code itself the tester may discover errors or assumptions not immediately obvious to someone treating the class as a black box.
Integration Testing	User Interface Testing The tester tests each interface function	Interface Design	For normal integration testing	<ul style="list-style-type: none"> Testing is done by moving through each and every menu item in the interface either in a top-down or bottom-up manner.
	Use-Case Testing The tester tests each use case	Use Cases	When the user interface is important	<ul style="list-style-type: none"> Testing is done by moving through each use case to ensure they work correctly. Usually combined with user interface testing because it does not test all interfaces.
System Testing	Interaction Testing Tests each process in a step-by-step fashion	Class Diagrams Sequence Diagrams Communication Diagrams	When the system performs data processing	<ul style="list-style-type: none"> The entire system begins as a set of stubs. Each class is added in turn and the results of the class compared to the correct result from the test data; when a class passes, the next class is added and the test rerun. This is done for each package. Once each package has passed all tests, then the process repeats integrating the packages.
	System Interface Testing Tests the exchange of data with other systems	Use-Case Diagram	When the system exchanges data	<ul style="list-style-type: none"> Because data transfers between systems are often automated and not monitored directly by the users it is critical to design tests to ensure they are being done correctly.
Acceptance Testing	Requirements Testing Tests to whether original business requirements are met	System Design, Unit Tests, and Integration Tests	For normal system testing	<ul style="list-style-type: none"> Ensures that changes made as a result of integration testing did not create new errors. Testers often pretend to be uninformed users and perform improper actions to ensure the system is immune to invalid actions (e.g., adding blank records).
	Usability Testing Tests how convenient the system is to use	Interface Design and Use Cases	When user interface is important	<ul style="list-style-type: none"> Often done by analyst with experience in how users think and in good interface design. Sometimes uses formal usability testing procedures discussed in Chapter 12.
Documentation Testing	Security Testing Tests disaster recovery & unauthorized access	Infrastructure Design	When the system is important	<ul style="list-style-type: none"> Security testing is a complex task, usually done by an infrastructure analyst assigned to the project. In extreme cases, a professional firm may be hired.
	Performance Testing Examines the ability to perform under high loads	System Proposal Infrastructure Design	When the system is important	<ul style="list-style-type: none"> High volumes of transactions are generated and given to the system. Often done by using special purpose testing software.
	Documentation Testing Tests the accuracy of the documentation	Help System, Procedures, Tutorials	For normal system testing	<ul style="list-style-type: none"> Analysts spot check or check every item on every page in all documentation to ensure the documentation items and examples work properly.
Alpha Testing	Alpha Testing Conducted by users to ensure they accept the system	System Tests	For normal acceptance testing	<ul style="list-style-type: none"> Often repeats previous tests but are conducted by users themselves to ensure they accept the system.
	Beta Testing Uses real data, not test data	No plan	When the system is important	<ul style="list-style-type: none"> Users closely monitor system for errors or useful improvements.

FIGURE 14-4 Types of Tests

**YOUR
TURN**
14-1 Test Planning for an ATM

Pretend you are a project manager for a bank developing software for ATMs. Develop a unit test plan for the user interface component of the ATM.

test plan that has a series of tests that in turn have tests. Integration testing is often done by a set of programmers and/or systems analysts.

From an object-oriented systems perspective, integration testing can be difficult. A single class can be in many different aggregations, because of the way in which objects can be combined to form new objects, class libraries, frameworks, components, and packages. Where is the best place to start the integration? Typically, the answer is to begin with the set of classes, or collaboration, used to support the highest priority use case (see Chapter 6). Also, dynamic binding makes it crucial to design the integration tests carefully to ensure that the combinations of methods are tested.

There are four approaches to integration testing: *user interface testing*, *use-case testing*, *interaction testing*, and *system interface testing* (see Figure 14-4). Most projects use all four approaches.

System Tests

System tests are usually conducted by the systems analysts to ensure that all classes work together without error. System testing is similar to integration testing but is much broader in scope. Whereas integration testing focuses on whether the classes work together without error, system tests examine how well the system meets business requirements (*requirements testing*), how convenient the system is to use (*usability testing*), how secure the system is (*security testing*), how well the system performs under a heavy load (*performance testing*), and how accurate the documentation of the system is (*documentation testing*). Figure 14-4 provides additional details on the different types of system tests.

CONCEPTS
IN ACTION
14-B Anatomy of a Friendly Hack

If you've seen the movie *Sneakers*, you know that there are professional security firms that organizations can hire to break in to their own networks to test security. BABank (a pseudonym) was about to launch a new online banking application, so it hired such a firm to test its security before the launch. The bank's system failed the security test—badly.

The security team began by mapping the bank's network. It used network security analysis software and dialing software to test for dial-in ports. The mapping process found a bunch of accounts with default passwords (i.e., passwords set by the manufacturer that are supposed to be changed when the systems are first set up). The team

then tricked several high-profile users into revealing their passwords to gain access to several high-privilege accounts. Once into these computers, the team used password cracking software to find passwords on these computers and ultimately gain the administrator passwords on several servers. At this point, the team transferred \$1,000 into their test account. They could have transferred more, but the security point was made.

Source: *Network World*, February 2, 1998.

Question

1. What was the value of the security tests in this case?

Acceptance Tests

Acceptance testing is done primarily by the users with support from the project team. The goal is to confirm that the system is complete, meets the business needs that prompted the system to be developed, and is acceptable to the users. Acceptance testing is done in two stages: *alpha testing*, in which users test the system using made-up data, and *beta testing*, in which users begin to use the system with real data but are carefully monitored for errors (see Figure 14-4).

DEVELOPING DOCUMENTATION

Like testing, developing documentation of the system must be done throughout the SDLC. There are two fundamentally different types of documentation: system documentation and user documentation. *System documentation* is intended to help programmers and systems analysts understand the application software and enable them to build it or maintain it after the system is installed. System documentation is largely a byproduct of the systems analysis and design process and is created as the project unfolds. Each step and phase produces documents that are essential in understanding how the system is constructed or is to be built. In many object-oriented development environments, it is possible to somewhat automate the creation of detailed documentation for classes and methods. For example, in Java, if the programmers use *javadoc*-style comments, it is possible to create HTML pages that document a class and its methods automatically by using the *javadoc* utility.¹⁰ Since most programmers look on documentation with much distaste, anything that can make documentation easier to create is useful.

User documentation (such as user's manuals, training manuals, and online help systems) is designed to help the user operate the system. Although most project teams expect users to have received training and to have read the user's manuals before operating the system, unfortunately this is not always the case. It is more common today—especially in the case of commercial software packages for microcomputers—for users to begin using the software without training or reading the user's manuals. In this section, we focus on user documentation.¹¹

User documentation is often left until the end of the project, which is a dangerous strategy. Developing good documentation takes longer than many people expect because it requires much more than simply writing a few pages. Producing documentation requires designing the documents (whether on paper or online), writing the text, editing them, and testing them. For good-quality documentation, this process usually takes about three hours per page (single-spaced) for paper-based documentation or two hours per screen for online documentation. Thus, a “simple” set of documentation such as a ten-page user's manual and a set of twenty help screens takes seventy hours. Of course, lower quality documentation can be produced faster.

The time required to develop and test user documentation should be built into the project plan. Most organizations plan for documentation development to start once the interface design and program specifications are complete. The initial draft of documentation is usually scheduled for completion immediately after the unit tests are complete. This reduces the chance that the documentation will need to be changed due to software changes (but doesn't eliminate it) and still leaves enough time for the documentation to be tested and revised before the acceptance tests are started.

¹⁰ For those who have used Java, *javadoc* is how the JDK documentation from Sun was created.

¹¹ For more information on developing documentation, see Thomas T. Barker, *Writing Software Documentation* (Boston, MA: Allyn and Bacon, 1998).

Although paper-based manuals are still important, online documentation is becoming more important. Paper-based documentation is simpler to use because it is more familiar to users, especially novices who have less computer experience; online documentation requires the users to learn one more set of commands. Paper-based documentation is also easier to flip through and gain a general understanding of its organization and topics and can be used far away from the computer itself.

There are four key strengths of online documentation that all but guarantee it will be the dominant form for the next century. First, searching for information is often simpler (provided the help search index is well designed) because the user can type in a variety of keywords to view information almost instantaneously rather than having to search through the index or table of contents in a paper document. Second, the same information can be presented several times in many different formats, so that the user can find and read the information in the most informative way (such redundancy is possible in paper documentation, but the cost and intimidating size of the resulting manual make it impractical). Third, online documentation enables many new ways for the user to interact with the documentation that are not possible in static paper documentation. For example, it is possible to use links or “tool tips” (i.e., pop-up text; see Chapter 12) to explain unfamiliar terms, and one can write “show-me” routines that demonstrate on the screen exactly what buttons to click and text to type. Finally, online documentation is significantly less expensive to distribute than paper documentation.

Types of Documentation

There are three fundamentally different types of user documentation: reference documents, procedures manuals, and tutorials. *Reference documents* (also called the help system) are designed to be used when the user needs to learn how to perform a specific function (e.g., updating a field, adding a new record). Often people read reference information when they have tried and failed to perform the function; writing reference documents requires special care because often the user is impatient or frustrated when he or she begins to read them.

Procedures manuals describe how to perform business tasks (e.g., printing a monthly report, taking a customer order). Each item in the procedures manual typically guides the user through a task that requires several functions or steps in the system. Therefore, each entry is typically much longer than an entry in a reference document.

Tutorials—obviously—teach people how to use major components of the system (e.g., an introduction to the basic operations of the system). Each entry in the tutorial is typically longer still than the entries in procedures manuals and is usually designed to be read in sequence (whereas entries in reference documents and procedures manuals are designed to be read individually).

Regardless of the type of user documentation, the overall process for developing it is similar to the process of developing interfaces (see Chapter 12). The developer first designs the general structure for the documentation and then develops the individual components within it.

Designing Documentation Structure

In this section, we focus on the development of online documentation because we believe it will become the most common form of user documentation. The general structure used in most online documentation, whether reference documents, procedures manuals, or tutorials is to develop a set of *documentation navigation controls* that lead the user to *documentation topics*. The documentation topics are the material the user wants to read, while the navigation controls are the way in which the user locates and accesses a specific topic.

Designing the structure of the documentation begins by identifying the different types of topics and navigation controls that need to be included. Figure 14-5 shows a commonly used structure for online reference documents (i.e., the help system). The documentation topics generally come from three sources. The first and most obvious source of topics is the set of commands and menus in the user interface. This set of topics is very useful if the user wants to understand how a particular command or menu is used.

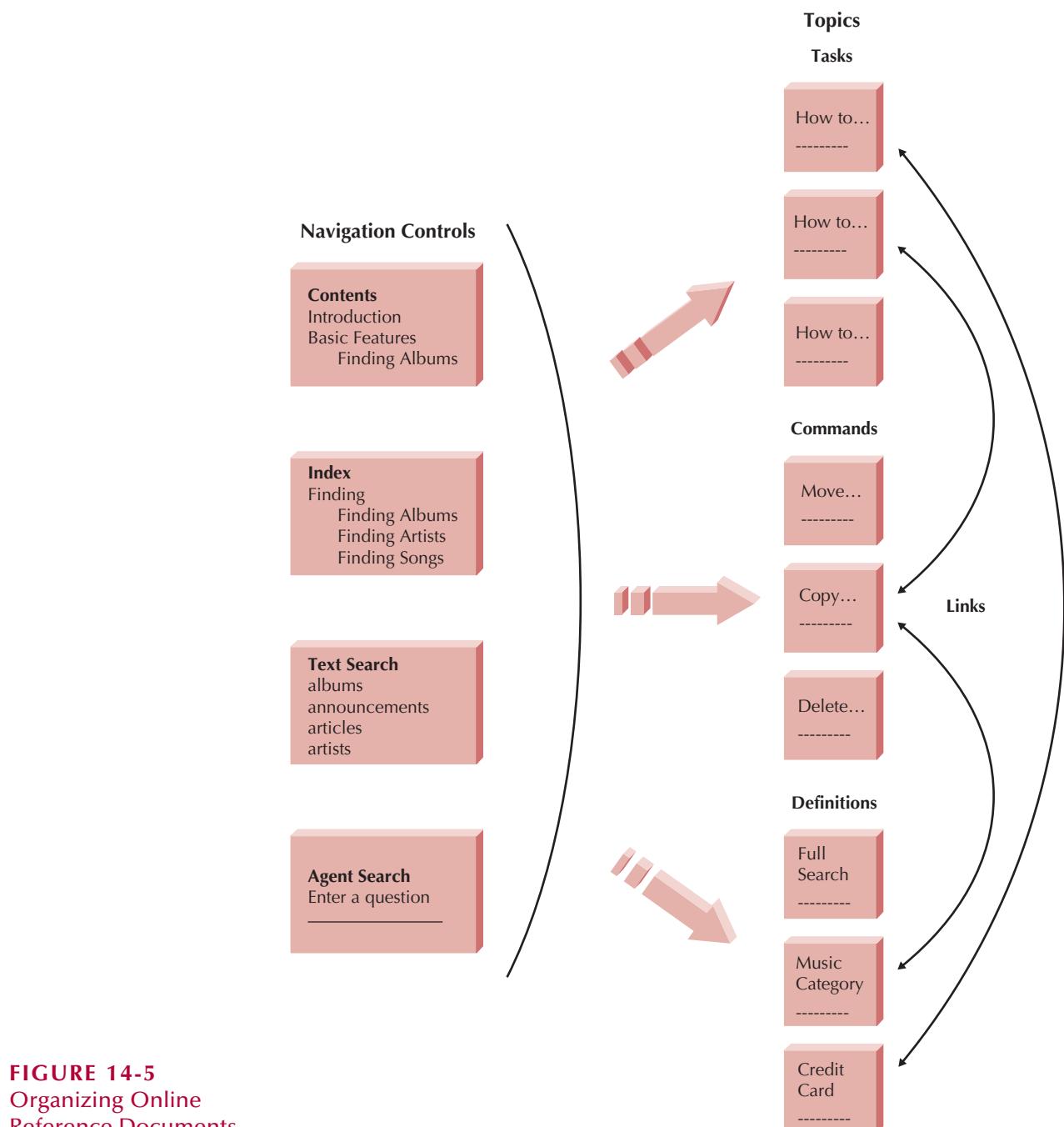


FIGURE 14-5
Organizing Online Reference Documents

The users often don't know what commands to look for or where they are in the system's menu structure. Instead, users have tasks they want to perform, and rather than thinking in terms of commands, they think in terms of their business tasks. Therefore, the second and often more useful set of topics focuses on how to perform certain tasks—usually those in the use scenarios, window navigation diagram, and the real use cases from the user interface design (see Chapter 12). These topics walk the user through the set of steps (often involving several keystrokes or mouse clicks) needed to perform some task.

The third set of topics is definitions of important terms. These terms are usually the use cases and classes in the system, but sometimes they also include commands.

There are five general types of navigation controls for topics, but not all systems use all five types (see Figure 14-5). The first is the table of contents that organizes the information in a logical form, as though the users were to read the reference documentation from start to finish.

The index provides access to the topics based on important keywords, in the same way that the index at the back of a book helps you find topics. Text search provides the ability to search through the topics either for any text the user types or for words that match a developer-specified set of words that is much larger than the words in the index. Unlike the index, text search typically provides no organization to the words (other than alphabetical). Some systems provide the ability to use an intelligent agent to help in the search (e.g., the Microsoft Office Assistant—also known as the paper clip guy). The fifth and final navigation control to topics are the Web-like links between topics that enable the user to click and move among topics.

Procedures manuals and tutorials are similar but often simpler in structure. Topics for procedures manuals usually come from the use scenarios, window navigation diagram, and the real use cases developed during interface design and from other basic tasks the users must perform. Topics for tutorials are usually organized around major sections of the system and the level of experience of the user. Most tutorials start with the basic, most commonly used commands and then move into more complex and less frequently used commands.

Writing Documentation Topics

The general format for topics is fairly similar across application systems and operating systems (see Figure 14-6). Topics typically start with very clear titles, followed by some introductory text that defines the topic, and then provide detailed, step-by-step instructions on how to perform what is being described (where appropriate). Many topics include screen images to help the user find items on the screen; some also have "show-me" examples in which the series of keystrokes and/or mouse movements and clicks needed to perform the function are demonstrated to the user. Most also include navigation controls to enable movement among topics, usually at the top of the window, plus links to other topics. Some also include links to "related topics" that include options or other commands and tasks the user may want to perform in concert with the topic being read.

Writing the topic content can be challenging. It requires a good understanding of the user (or more accurately the range of users) and a knowledge of what skills the user(s) currently have and can be expected to import from other systems and tools they are using or have used (including the system the new system is replacing). Topics should always be written from the viewpoint of the user and describe what the user wants to accomplish, not what the system can do. Figure 14-7 provides some general guidelines to improve the quality of documentation text.¹²

¹² One of the best books to explain the art of writing is William Strunk and E.B. White, *Elements of Style*, 4th ed. (Boston: Allyn and Bacon, 2000).

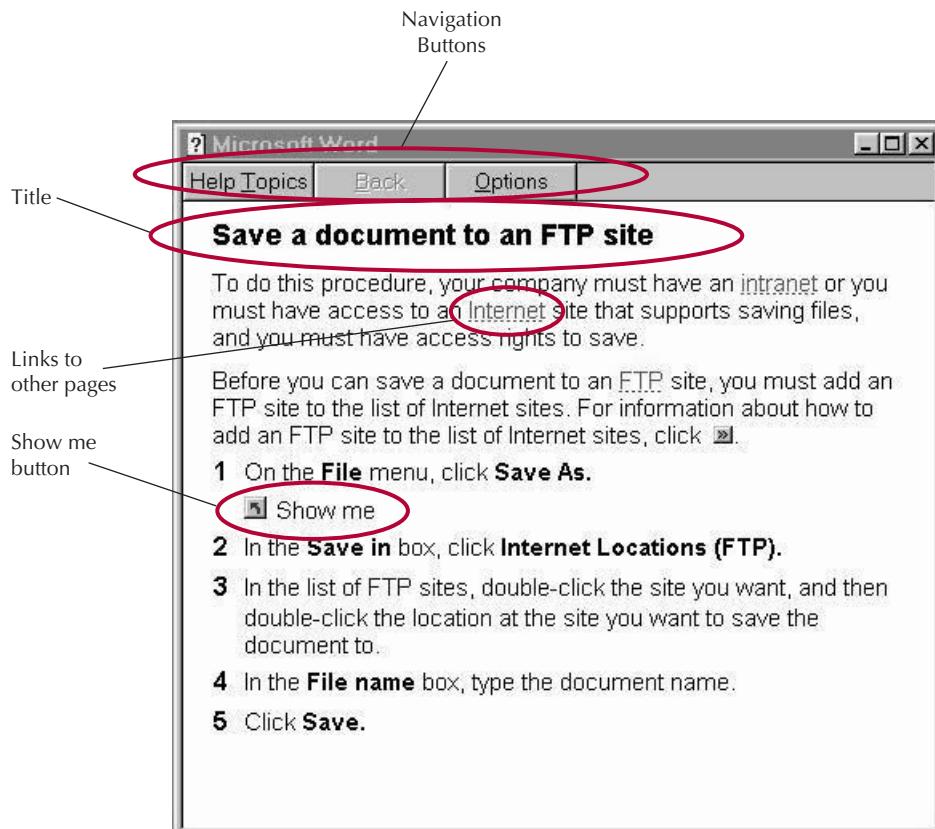


FIGURE 14-6
A Help Topic in
Microsoft Word

Identifying Navigation Terms

As you write the documentation topics you also begin to identify the terms that will be used to help users find topics. The table of contents is usually the most straightforward because it is developed from the logical structure of the documentation topics, whether reference topics, procedure topics, or tutorial topics. The items for the index and search engine require more care because they are developed from the major parts of the system and the users' business functions. Every time you write a topic, you must also list the terms that will be used to find the topic. Terms for the index and search engine can come from four distinct sources.

The first source is the set of the commands in the user interface, such as open file, modify customer, and print open orders. All commands contain two parts (action and object). It is important to develop the index for both parts because users could search for information using either part. A user looking for more information about saving files, for example, might search by using the term "save" or the term "files."

The second source is the set of major concepts in the system, which are often use cases and classes. In the case of CD Selections, for example, this might include music category, album, and shipping costs.

A third source is the set of business tasks the user performs, such as ordering replacement unit or making an appointment. Often these will be contained in the command set, but sometimes they require several commands and use terms that always appear in the system. Good sources for these terms are the use scenarios and real use cases developed during interface design (see Chapter 12).

A fourth, often controversial, source is the set of synonyms for the three sets of items above. Users sometimes don't think in terms of the nicely defined terms used by the system. They may try to find information on how to "stop" or "quit" rather than "exit," or "erase" rather than "delete." Including synonyms in the index increases the complexity and size of the documentation system but can greatly improve the usefulness of the system to the users.

Guideline	Before the Guideline	After the Guideline
Use the active voice: The active voice creates more active and readable text by putting the subject at the start of the sentence, the verb in the middle, and the object at the end.	Finding albums is done using the album title, the artist's name, or a song title.	You can find an album by using the album title, the artist's name, or a song title.
Use e-prime style: E-prime style creates more active writing by omitting all forms of the verb to be.	The text you want to copy must be selected before you click on the copy button.	Select the text you want to copy before you click on the copy button.
Use consistent terms: Always use the same term to refer to the same items, rather than switching among synonyms (e.g., change, modify, update).	Select the text you want to copy. Pressing the copy button will copy the marked text to the new location.	Select the text you want to copy. Pressing the copy button will copy the selected text to the new location.
Use simple language: Always use the simplest language possible to accurately convey the meaning. This does not mean you should "dumb down" the text but that you should avoid artificially inflating its complexity. Avoid separating subjects and verbs and try to use the fewest words possible. (When you encounter a complex piece of text, try eliminating words; you may be surprised at how few words are really needed to convey meaning.)	The Georgia Statewide Academic and Medical System (GSAMS) is a cooperative and collaborative distance learning network in the state of Georgia. The organization in Atlanta that administers and manages the technical and overall operations of the currently more than 300 interactive audio and video teleconferencing classrooms throughout Georgia system is the Department of Administrative Service (DOAS). (56 words)	The Department of Administrative Service (DOAS) in Atlanta manages the Georgia Statewide Academic and Medical System (GSAMS), a distance learning network with more than 300 teleconferencing classrooms throughout Georgia. (29 words)
Use friendly language: Too often, documentation is cold and sterile because it is written in a very formal manner. Remember, you are writing for a person, not a computer.	Blank disks have been provided to you by Operations. It is suggested that you ensure your data is not lost by making backup copies of all essential data.	You should make a backup copy of all data that is important to you. If you need more diskettes, contact Operations.
Use parallel grammatical structures: Parallel grammatical structures indicate the similarity among items in list and help the reader understand content.	Opening files Saving a document How to delete files	Opening a file Saving a file Deleting a file
Use steps correctly: Novices often interperse action and the results of action when describing a step-by-step process. Steps are always actions.	1. Press the customer button. 2. The customer dialogue box will appear. 3. Type the customer ID and press the submit button and the customer record will appear.	1. Press the customer button. 2. Type the customer ID in the customer dialogue box when it appears. 3. Press the submit button to view the customer record for this customer.
Use short paragraphs: Readers of documentation usually quickly scan text to find the information they need, so the text in the middle of long paragraphs is often overlooked. Use separate paragraphs to help readers find information more quickly.		

Source: Adapted from T. T. Barker, *Writing Software Documentation* (Boston: Allyn & Bacon, 1998).

FIGURE 14-7 Guidelines for Crafting Documentation Topics

**YOUR
TURN**
14-2 Documentation for an ATM

Pretend you are a project manager for a bank developing software for ATMs. Develop the part of an online help system that deals with withdrawing and depositing cash. Be

sure to include the different types of accounts, such as checking and savings.

APPLYING THE CONCEPTS AT CD SELECTIONS

Managing Programming

Three programmers were assigned by CD Selections to develop the three major parts of the Internet sales system. The first was the Web interface, both the client side (browser) and the server side. The second was the client–server-based management system (managing the CD information and marketing materials databases). The third was the interfaces between the Internet sales system and CD Selections' existing distribution system and the credit card center. Programming went smoothly and despite a few minor problems proceeded according to plan.

Testing

While the programmers were working, Alec began developing the test plans and user documentation. The test plans for the three components were similar but slightly more intensive for the Web interface component (see Figure 14-8). Unit testing would use black-box testing based on the CRC cards, class diagrams, and contracts for all components. Figure 14-9 shows part of the class-invariant test specification for the Order class in the Web interface component.

Integration testing for the Web interface and system management component would be subjected to all user interface and use-case tests to ensure that the interface would work properly. The system interface component would undergo system interface tests to ensure that the system performed calculations properly and was capable of exchanging data with the CD Selections' other systems and the credit card center.

Systems tests are by definition tests of the entire system—all components together. However, not all parts of the system would receive the same level of testing. Requirements tests would be conducted on all parts of the system to ensure that all requirements were met. Because security was a critical issue, the security of all aspects of the system would be tested. Security tests would be developed by CD Selections' infrastructure team, and once the system passed those tests, an external security consulting firm would be hired to attempt to break in to the system.

Performance was an important issue for the parts of the system used by the customer (the Web interface and the system interfaces to the credit card and inventory systems) but not as important for the management component that would be used by staff,

Test Stage	Web Interface	System Management	System Interfaces
Unit tests	Black-box tests	Black-box tests	Black-box tests
Integration tests	User interface tests; use-case tests	User interface tests; use-case tests	System interface tests
System tests	Requirements tests; security tests; performance tests; usability tests	Requirements tests; security tests	Requirements tests; security tests; performance tests
Acceptance tests	Alpha test; beta test	Alpha test; beta test	Alpha test; beta test

FIGURE 14-8
CD Selections'
Test Plan

Class Invariant Test Specification					
Page <u>5</u> of <u>15</u>					
Class Name: <u>Order</u>		Version Number: <u>3</u>		CRC Card ID: <u>15</u>	
Tester: <u>Susan Doe</u>		Date Designed: <u>9/9</u>		Date Conducted: _____	
Testing Objectives: Ensure that the information entered by the customer on the place order form is valid.					
Test Cases					
Invariant Description	Original Attribute Value	Event	New Attribute Value	Expected Result	Result P/F
Attribute Name: CD Number					
1) <u>(1..1)</u>	<u>Null</u>	CreateOrder	<u>Null</u>	<u>F</u>	_____
2) <u>(unsigned long)</u>	<u>Null</u>	CreateOrder	<u>ABC</u>	<u>F</u>	_____
3) <u>(unsigned long)</u>	<u>Null</u>	CreateOrder	<u>123</u>	<u>P</u>	_____
Attribute Name: CD Name					
1) <u>(1..1)</u>	<u>Null</u>	CreateOrder	<u>Null</u>	<u>F</u>	_____
2) <u>(String)</u>	<u>Null</u>	CreateOrder	<u>ABC</u>	<u>P</u>	_____
3) <u>(String)</u>	<u>Null</u>	CreateOrder	<u>123</u>	<u>P</u>	_____
Attribute Name: Artist Name					
1) <u>(1..1)</u>	<u>Null</u>	CreateOrder	<u>Null</u>	<u>F</u>	_____
2) <u>(String)</u>	<u>Null</u>	CreateOrder	<u>ABC</u>	<u>P</u>	_____
3) <u>(String)</u>	<u>Null</u>	CreateOrder	<u>123</u>	<u>P</u>	_____

FIGURE 14-9
Partial Class Invariant Test Specification for the Order Class

not customers. The customer-facing components would undergo rigorous performance testing to see how many transactions (whether searching or purchasing) they could handle before they were unable to provide a response time of two seconds or less. Alec also developed an upgrade plan so that as demand on the system increased, there was a clear plan for when and how to increase the processing capability of the system.

Finally, formal usability tests would be conducted on the Web interface portion of the system with six potential users (both novice and expert Internet users).

Acceptance tests would be conducted in two stages, alpha and beta. Alpha tests would be done during the training of CD Selections' staff. The Internet Sales Group manager would work together with Alec to develop a series of tests and training exercises to train the Internet Sales Group staff in how to use the system. They would then load the real CD data into the system and begin adding marketing materials. These same staff and other CD Selections staff members would also pretend to be customers and test the Web interface.

Beta testing would be done by “going live” with the Web site but its existence would only be announced to CD Selections’ employees. As an incentive to try the Web site (rather than buying from the store in which they worked), employees would be offered triple their normal employee discount for all products ordered from the Web site. The site would also have a prominent button on every screen that would enable employees to e-mail comments to the project team, and the announcement would encourage employees to report problems, suggestions, and compliments to the project team. After one month, assuming all went well, the beta test would be completed, and the Internet sales site would be linked to the main Web site and advertised to the general public.

Developing User Documentation

Three types of documentation (reference documents, procedures manuals, and tutorials) could be produced for the Web interface and the management component. Since the number of CD Selections' staff using the system management component would be small, Alec decided to produce only the reference documentation (an online help system). He felt that an intensive training program and a one-month beta test period would be sufficient without tutorials and formal procedures manuals. Similarly, he felt that the process of ordering CDs and the Web interface itself was simple enough to not require a tutorial on the Web—a help system would be sufficient (and a procedures manual didn't make sense).

Alec decided that the reference documents for both the Web interface and system management components would contain help topics for user tasks, commands, and definitions. He also decided that the documentation component would contain four types of navigation controls: a table of contents, an index, a find, and links to definitions. He did not feel that the system was complex enough to benefit from a search agent.

After these decisions were made, Alec delegated the development of the reference documents to a technical writer assigned to the project team. Figure 14-10 shows examples of a few of topics that the writer developed. The tasks and commands were taken directly from the interface design. The list of definitions was developed once the tasks and commands were developed based on the writer's experience in understanding what terms might be confusing to the user.

Once the topic list was developed, the technical writer then began writing the topics themselves as well as the navigation controls to access. Figure 14-11 shows an example of one topic taken from the task list: how to place an order. This topic presents a brief description of what it is and then leads the user through the step-by-step process needed to complete the task. The topic also lists the navigation controls that will be used to find the topic in terms of the table of contents entries, the index entries, and search entries. It also lists what words in the topic itself will have links to other topics (e.g., shopping cart).

SUMMARY

Managing Programming

Programming is done by programmers, so systems analysts have few responsibilities during this stage. The project manager, however, is usually very busy. The first task is to assign the programmers, ideally the fewest possible to complete the project because coordination problems increase as the size of the programming team increases. Coordination can be improved by having regular meetings, ensuring that standards are followed, implementing change control, and using CASE tools effectively. One of the key functions of the project manager is to manage the schedule and adjust it for delays. Two common causes of delays are scope creep and minor slippages that go unnoticed.

Tasks	Commands	Terms
Find an album	Find	Album
Add an album to my shopping cart	Browse	Artist
Placing an order	Quick search	Music type
How to buy	Full search	Special deals
What's in my shopping cart?		Cart
		Shopping cart

FIGURE 14-10
Sample Help Topics for
CD Selections

FIGURE 14-11
Example Documentation Topic for CD Selections

The screenshot shows a help topic titled "How to Place an Order". The main content area contains the following text:

When you are ready to pay for the merchandise you have selected (the items in your shopping cart) you can place your order. There are four steps.

1. Move to the Place order Page

Click on the **Place order** button to move to the place order page.

2. Make sure you are ordering what you want

The place order screen displays all the items in your shopping cart. Read through the list to make sure these are what you want because once you submit your credit card information you cannot change the order.

you can delete an item by

On the right side, there is a sidebar titled "Navigation Controls" with the following sections:

- Table of Contents** list: How to Place an Order
- Index** list: Credit Card, Order, Pay, Place order
- Search** find by: Credit Card, Delete Items, Order, Pay, Place order, Shopping Cart, Verify Order
- Links:** Shopping Cart

Test Planning

Tests must be carefully planned because the cost of fixing one major bug after the system is installed can easily exceed the annual salary of a programmer. A test plan contains several tests that examine different aspects of the system. A test, in turn, specifies several test cases that will be examined by the testers. A unit test examines a class within the system; test cases come from the class specifications or the class code itself. An integration test examines how well several classes work together; test cases come from the interface design, use cases, the use case diagram, sequence diagrams, and communication diagrams. A system test examines the system as a whole and is broader than the unit and integration tests; test cases come from the system design, the infrastructure design, and the unit and integration tests. Acceptance testing is done by the users to determine whether the system is acceptable to them; it draws on the system test plans (alpha testing) and on the real work the users perform (beta testing).

Documentation

Documentation, both user documentation and system documentation, is moving away from paper-based documents to online documentation. There are three types of user documentation: reference documents are designed to be used when the user needs to learn how to perform a specific function (e.g., an online help system), procedures manuals describe how to perform business tasks, while tutorials teach people how to use the system. Documentation navigation controls (e.g., a table of contents, index, find, intelligent agents, or links between pages) enable users to find documentation topics (e.g., how to perform a function, how to use an interface command, an explanation of a term).

KEY TERMS

Acceptance test
Alpha test
Beta test
Black-box testing

Change control
Construction
Documentation navigation control
Documentation testing

Documentation topic
Hardcoded value
Integration test
Interaction testing

Performance testing	Slippage	Test specification
Procedures manual	Stub	Tutorial
Program log	System documentation	Unit test
Reference document	System interface testing	Usability testing
Requirements testing	System test	Use-case testing
Risk assessment	Test	User documentation
Scope creep	Test case	User interface testing
Security testing	Test plan	White-box testing

QUESTIONS

1. Why is testing important?
2. What is the primary role of systems analysts during the programming stage?
3. In *The Mythical Man-Month*, Frederick Brooks argues that adding more programmers to a late project makes it later. Why?
4. What is the purpose of testing?
5. Describe how object-orientation impacts testing.
6. Compare and contrast the terms test, test plan, and test case.
7. What is a stub, and why is it used in testing?
8. What is the primary goal of unit testing?
9. Compare and contrast black-box testing and white-box testing.
10. How are the test cases developed for unit tests?
11. What are the different types of unit tests?
12. What is the primary goal of integration testing?
13. How are the test cases developed for integration tests?
14. What is the primary goal of system testing?
15. How are the test cases developed for system tests?
16. What is the primary goal of acceptance testing?
17. How are the test cases developed for acceptance tests?
18. Compare and contrast alpha testing and beta testing.
19. Compare and contrast user documentation and system documentation.
20. Why is online documentation becoming more important?
21. What are the primary disadvantages of online documentation?
22. Compare and contrast reference documents, procedures manuals, and tutorials.
23. What are five types of documentation navigation controls?
24. What are the commonly used sources of documentation topics? Which is the most important? Why?
25. What are the commonly used sources of documentation navigation controls? Which is the most important? Why?

EXERCISES

- A. Create an invariant test specification for the class you chose for the Health Club in Exercise A in Chapter 10.
- B. Create an invariant test specification for the class you chose for the Picnics R Us in Exercise C in Chapter 10.
- C. Create an invariant test specification for the class you chose for the Of-The-Month Club (OTMC) in Exercise E in Chapter 10.
- D. Create an invariant test specification for the class you chose for the Library in Exercise G in Chapter 10.
- E. If the registration system at your university does not have a good online help system, develop an online help system for one screen of the user interface.
- F. Examine and prepare a report on the online help system for the calculator program in Windows (or a similar on the Mac or Unix). (You will likely be surprised at the amount of help for such a simple program.)
- G. Compare and contrast the online help at two different Web sites that enable you to perform some function (e.g., make travel reservations, order books).

MINICASES

1. Pete is a project manager on a new systems development project. This project is Pete's first experience as a project manager, and he has led his team successfully to the programming phase of the project. The project

has not always gone smoothly, and Pete has made a few mistakes, but he is generally pleased with the progress of his team and the quality of the system being developed. Now that programming has begun,

Pete has been hoping for a little break in the hectic pace of his workday.

Prior to beginning programming, Pete recognized that the time estimates made earlier in the project were too optimistic. However, he was firmly committed to meeting the project deadline because of his desire for his first project as project manager to be a success. In anticipation of this time pressure problem, Pete arranged with the Human Resources department to bring in two new college graduates and two college interns to beef up the programming staff. Pete would have liked to find some staff with more experience, but the budget was too tight, and he was committed to keeping the project budget under control.

Pete made his programming assignments, and work on the programs began about two weeks ago. Now, Pete has started to hear some rumbles from the programming team leaders that may signal trouble. It seems that the programmers have reported several instances where they wrote programs, only to be unable to find them when they went to test them. Also, several programmers have opened programs that they had written, only to find that someone had changed portions of their programs without their knowledge.

- a. Is the programming phase of a project a time for the project manager to relax? Why or why not?
 - b. What problems can you identify in this situation? What advice do you have for the project manager? How likely does it seem that Pete will achieve his desired goals of being on time and within budget if nothing is done?
2. The systems analysts are developing the test plan for the user interface for the Holiday Travel Vehicles system. As the salespeople are entering a sales invoice into the system, they will be able to either enter an option code into a text box, or select an option code from a drop-down list. A combo box was used to implement this, since it was felt that the salespeople would quickly become familiar with the most common option codes and would prefer entering them directly to speed up the entry process.

It is now time to develop the test for validating the option code field during data entry. If the customer did not request any dealer-installed options for the vehicle, the salesperson should enter “none”; the field should not be blank. The valid option codes are four-character alphabetic codes and should be matched against a list of valid codes.

Prepare a test plan for the test of the option code field during data entry.

PART FOUR

IMPLEMENTATION PHASE

During the implementation phase, the actual system is built. Building a successful information system requires a set of activities: programming, testing, and documenting the system. Installing an information system requires switching from the current system to the new system. This conversion process can be quite involved. Not only does it involve shutting the old system down and turning the new one on, it also can involve a significant training effort. Finally, operating the system may uncover additional requirements that can be fed back into the next build.

CHAPTER 14

CONSTRUCTION

This chapter discusses the activities needed to successfully build the information system: programming, testing, and documenting the system. Programming is time-consuming and costly, but except in unusual circumstances, it is the simplest for the systems analyst because it is well understood. For this reason, the system analyst focuses on testing (proving that the system works as designed) and developing documentation during this part of the SDLC.

OBJECTIVES

- Be familiar with the system construction process.
- Understand different types of tests and when to use them.
- Understand how to develop documentation.

CHAPTER OUTLINE

Introduction	Acceptance Tests
Managing Programming	Developing Documentation
Assigning Programmers	Types of Documentation
Coordinating Activities	Designing Documentation Structure
Managing the Schedule	Writing Documentation Topics
Designing Tests	Identifying Navigation Terms
Testing and Object-Orientation	Applying the Concepts at CD Selections
Test Planning	Managing Programming
Unit Tests	Testing
Integration Tests	Developing User Documentation
System Tests	Summary

INTRODUCTION

When people first learn about developing information systems, usually they immediately think about writing programs. Programming can be the largest single component of any systems development project in terms of both time and cost. However, it also can be the best understood component and therefore—except in rare circumstances—offers the least problems of all aspects of the SDLC. When projects fail, it is usually not because the programmers were unable to write the programs but because the analysis, design, installation, and/or project management were done poorly. In this chapter, we focus on the construction and testing of the software and the documentation.

Construction is the development of all parts of the system, including the software itself, documentation, and new operating procedures. Programming is often seen as the focal point of system development. After all, system development *is* writing programs. It is the reason why we do all the analysis and design. And it's fun. Many beginning programmers see testing and documentation as bothersome afterthoughts. Testing and documentation aren't fun, so they often receive less attention than the creative activity of writing programs.

Programming and testing, however, are very similar to writing and editing. No professional writer (or student writing an important term paper) would stop after writing the first draft. Rereading, editing, and revising the initial draft into a good paper is the hallmark of good writing. Likewise, thorough testing is the hallmark of professional software developers. Most professional organizations devote more time and money to testing (and the subsequent revision and retesting) than to writing the programs in the first place.

The reasons are simple economics: downtime and failures caused by software bugs¹ are extremely expensive. Many large organizations estimate the costs of downtime of critical applications at \$50,000 to \$200,000 *per hour*.² One serious bug that causes an hour of downtime can cost more than one year's salary of a programmer—and how often are bugs found and fixed in one hour? Testing is therefore a form of insurance. Organizations are willing to spend a lot of time and money to prevent the possibility of major failures after the system is installed.

Therefore, programs are not usually considered finished until the test for that program is passed. For this reason, programming and testing are tightly coupled, and since programming is the primary job of the programmer (not the analyst), testing (not programming) often becomes the focus of the construction stage for the systems analysis team.

In this chapter, we discuss three parts of the construction step: programming, testing, and writing the documentation. Because programming is primarily the job of programmers, not systems analysts, and because this is not a programming book, we devote less time to programming than to testing and documentation.

MANAGING PROGRAMMING

In general, systems analysts do not write programs; programmers write programs. Therefore, the primary task of the systems analysts during programming is ... waiting. However, the project manager is usually very busy *managing* the programming effort by assigning the programmers, coordinating the activities, and managing the programming schedule.³

Assigning Programmers

The first step in programming is assigning modules to the programmers. As discussed in Chapter 10, each module (class, object, or method) should be as separate and distinct as possible from the other modules. The project manager first groups together classes that are related so that each programmer is working on related classes. These groups of classes are then assigned to programmers. A good place to start is to look at the package diagrams.

¹ When I was an undergraduate, I had the opportunity to hear Admiral Grace Hopper tell how the term bug was introduced. She was working on one of the early Navy computers when suddenly it failed. The computer would not restart properly so she began to search for failed vacuum tubes. She found a moth inside one tube and recorded in the log book that a bug had caused the computer to crash. From then on, every computer crash was jokingly blamed on a bug (as opposed to programmer error), and eventually the term bug entered the general language of computing.

² See Billie Shea, "Quality Patrol: Eye on the Enterprise," *Application Development Trends* (November 5, 1998): 31–38.

³ One of the best books on managing programming (even though it was first written 25 years ago) is that by Frederick P. Brooks, Jr., *The Mythical Man-Month*, 20th Anniversary edition (Reading, MA: Addison-Wesley, 1995).

CONCEPTS**IN ACTION****14-A The Cost of a Bug**

My first programming job in 1977 was to convert a set of application systems from one version of COBOL to another version of COBOL for the government of Prince Edward Island. The testing approach was to first run a set of test data through the old system and then run it through the new system to ensure that the results from the two matched. If they matched, then the last three months of production data were run through both to ensure that they too matched.

Things went well until I began to convert the Gas Tax system that kept records on everyone authorized to purchase gasoline without paying tax. The test data ran fine, but the results using the production data were peculiar. The old and new systems matched, but rather than listing several thousand records, the report listed only fifty. I checked the production data file and found it listed only fifty records, not the thousands that were supposed to be there.

The system worked by copying the existing gas tax records file into a new file and making changes in the new file. The old file was then copied to tape backup. There was a bug in the program such that if there were no changes to the file, a new file was created, but no records were copied into it.

I checked the tape backups and found one with the full set of data that was scheduled to be overwritten three days after I discovered the problem. The government was only three days away from losing all gas tax records.

—Alan Dennis

Question

1. What would have been the cost of this bug if it hadn't been caught?

One rule of system development is that the more programmers who are involved in a project, the longer the system will take to build. This is because as the size of the programming team increases, the need for coordination increases exponentially, and the more coordination that is required, the less time programmers can spend actually writing systems. The best size is the smallest possible programming team. When projects are so complex that they require a large team, the best strategy is to try to break the project into a series of smaller parts that can function as independently as possible.

Coordinating Activities

Coordination can be done through both high-tech and low-tech means. The simplest approach is to have a weekly project meeting to discuss any changes to the system that have arisen during the past week—or just any issues that have come up. Regular meetings, even if they are brief, encourage the widespread communication and discussion of issues before they become problems.

Another important way to improve coordination is to create and follow standards that can range from formal rules for naming files to forms that must be completed when goals are reached to programming guidelines (see Chapter 4). When a team forms standards and then follows them, the project can be completed faster because task coordination is less complex.

The analysts also must put mechanisms in place to keep the programming effort well-organized. Many project teams set up three “areas” in which programmers can work: a development area, a testing area, and a production area. These areas can be different directories on a server hard disk, different servers, or different physical locations, but the point is that files, data, and programs are separated based on their status of completion. At first, programmers access and build files within the development area and then copy them to the testing area when the programmers are “finished.” If a program does not pass a test, it is sent back to development. Once all programs and the like are

tested and ready to support the new system, they are copied into the production area—the location where the final system will reside.

One of the more important things to coordinate is the traceability of the implementation of the system back to the original requirements. Each class and method must be associated with an individual requirement or a set of requirements. In most object-oriented system development approaches today capture and document the requirements in use cases. As such, it is critical to guarantee that all classes and methods can be traced back to an individual use case or a set of use cases. This is why in our approach (MOOSAD), we have included a section for “Associated Use Cases” on the CRC cards (see Figures 7-1 and 10-11) and the contracts (see Figure 10-13).

Keeping files and programs in different places based on completion status helps manage *change control*, the action of coordinating a system as it changes through construction. Another change control technique is keeping track of which programmer changes which classes and packages by using a *program log*. The log is merely a form on which programmers “sign-out” classes and packages to write and “sign-in” when they are completed. Both the programming areas and program log help the analysts understand exactly who has worked on what and to determine the system’s current status. Without these techniques, files can be put into production without the proper testing; two programmers, for example, could start working on the same class or package at the same time.

If a CASE tool is used during the construction step, it can be very helpful for change control because many CASE tools are set up to track the status of programs and help manage programmers as they work. In most cases, maintaining coordination is not conceptually complex. It just requires a lot of attention and discipline to track small details.

Managing the Schedule

The time estimates that were produced during the initial planning phase and refined during the analysis and design phases will almost always need to be refined as the project progresses during construction because it is virtually impossible to develop an exact assessment of the project’s schedule. As we discussed in Chapter 4, a well-done set of time estimates will usually have a 10 percent margin of error by the time you reach the construction step. It is critical that the time estimates be revised as the construction step proceeds. If a program module takes longer to develop than expected, then the prudent response is to move the expected completion date later by the same amount.

One of the most common causes of schedule problems is scope creep. *Scope creep* occurs when new requirements are added to the project after the system design was finalized. Scope creep can be very expensive because changes made late in the SDLC can require much of the completed system design (and even programs already written) to be redone. Any proposed change during the construction phase must require the approval of the project manager and should only be done after a quick cost-benefit analysis has been done.

Another common cause is the unnoticed day-by-day *slippage* in the schedule. One package is a day later here, another one a day late there. Pretty soon these minor delays add up, and the project is noticeably behind schedule. Once again, the key to managing the programming effort is to watch these minor slippages carefully and update the schedule accordingly.

Typically, a project manager will create a *risk assessment* that tracks potential risks along with an evaluation of their likelihood and potential impact. As the construction step moves to a close, the list of risks will change as some items are removed and others surface. The best project managers, however, work hard to keep risks from having an impact on the schedule and costs associated with the project.

PRACTICAL**TIP**

In previous chapters, we discussed classic mistakes and how to avoid them. Here, we summarize four classic mistakes in the implementation phase.⁴

1. Research-oriented Development: Using state-of-the-art technology requires research-oriented development that explores the new technology because “bleeding edge” tools and techniques are not well understood, are not well documented, and do not function exactly as promised.

Solution: If you use state-of-the-art technology, you need to significantly increase the project’s time and cost estimates even if (some experts would say *especially if*) such technologies claim to reduce time and effort.

2. Using Low-Cost Personnel: You get what you pay for. The lowest cost consultant or the staff member is significantly less productive than the best staff. Several studies have shown that the best programmers produce software six to eight times faster than the least productive (yet cost only 50 to 100 percent more).



Solution: If cost is a critical issue, assign the best, most expensive personnel; never assign entry-level personnel in an attempt to save costs.

3. Lack of Code Control: On large projects, programmers need to coordinate changes to the program source code (so that two programmers don’t try to change the same program at the same time and overwrite the other’s changes). Although manual procedures appear to work (e.g., sending e-mail notes to others when you work on a program to tell them not to), mistakes are inevitable.

Solution: Use a source code library that requires programmers to “check out” programs and prohibits others from working on them at the same time.

4. Inadequate Testing: The number one reason for project failure during implementation is ad hoc testing—where programmers and analysts test the system without formal test plans.

Solution: Always allocate sufficient time in the project plan for formal testing.

DESIGNING TESTS⁵

In object-oriented systems, the temptation is to minimize testing. After all, through the use of patterns, frameworks, class libraries, and components, much of the system has been tested previously. Therefore, we should not have to test as much. Right? Wrong!! Testing is more critical to object-oriented systems than to systems developed in the past. Based on encapsulation (and information hiding), polymorphism (and dynamic binding), inheritance, and reuse, thorough testing is much more difficult and critical. Testing must be done systematically and the results documented so that the project team knows what has and has not been tested.

The purpose of testing is not to demonstrate that the system is free of errors. Indeed, it is not possible to prove that a system is error free, especially object-oriented systems. This is similar to theory testing. You cannot prove a theory. If a test fails to find problems with a theory, your confidence in the theory is increased. However, if a test succeeds in finding a problem, then the theory has been falsified. Software testing is similar in that it can only show the existence of errors. As such, the purpose of testing is to uncover as many errors as feasible.⁶

⁴ Adapted from Steve McConnell, *Rapid Development* (Redmond, WA: Microsoft Press, 1996).

⁵ This section is based on material contained in Imran Bashir and Amrit L. Goel, *Testing Object-Oriented Software: Life Cycle Solutions* (New York: Springer Verlag, 1999); Bernd Bruegge and Allen H. Dutoit, *Object-Oriented Software Engineering: Conquering Complex and Changing Systems*, (Englewood Cliffs, NJ: Prentice Hall, 2000); Philippe Kruchten, *The Rational Unified Process: An Introduction*, 2nd ed., (Reading, MA: Addison-Wesley, 2000); and John D. McGregor and David A. Sykes, *A Practical Guide to Testing Object-Oriented Software* (Reading, MA: Addison-Wesley, 2001). For more information on testing object-oriented software, see Robert V. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools* (Reading, MA: Addison-Wesley, 1999); and Shel Sieget, *Object-Oriented Software Testing: A Hierarchical Approach* (New York: Wiley, 1996).

⁶ It is not cost-effective to get each and every error out of the software. Except in simple examples, it is in fact impossible. There are simply too many combinations to check.

There are four general stages of tests: unit tests, integration tests, system tests, and acceptance tests. Although each application system is different, most errors are found during integration and system testing (see Figure 14-1).

In each of the following sections, we describe the four stages. However, before doing this, we describe the effect that the object-oriented characteristics have on testing and the necessary planning and management activities that must take place to have a successful testing program.

Testing and Object-Orientation

Most testing techniques have been developed to support nonobject-oriented development. Object-oriented development is still relatively new. As such, most of the testing approaches have had to be adapted to object-oriented systems. The characteristics of object-oriented systems that affect testing the most are encapsulation (and information hiding), polymorphism (and dynamic binding), inheritance, and the use of patterns, class libraries, frameworks, and components. Also, the sheer volume of products that come out of a typical object-oriented development process has increased the importance of testing in object-oriented development.

Encapsulation and Information Hiding Encapsulation and information hiding allow processes and data to be combined to create holistic entities (i.e., objects). Furthermore, they support hiding everything behind a visible interface. Although this allows the system to be modified and maintained in an effective and efficient manner, it makes testing the system problematic. What do you need to test to build confidence in the system's ability to meet the user's need? Answer, you need to test the business process that is represented in the use cases. However, the business process is distributed over a set of collaborating classes and contained in the methods of those classes. The only way to know the effect that a business process has on a system is to look at the state changes that take place in the system. But in object-oriented systems, the instances of the classes hide the data behind a class boundary. How is it possible then to see the impact of a business process?

A second issue raised by encapsulation and information hiding is the definition of a "unit" for unit testing. What is the unit to be tested? Is it the package, class, or method? In

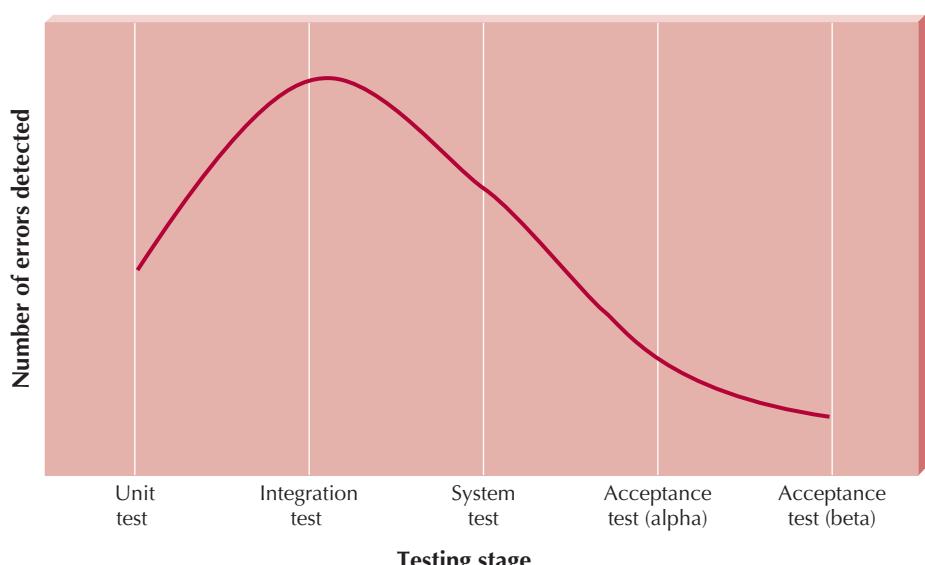


FIGURE 14-1
Error discovery rates for different stages of tests

traditional approaches, the answer would be the process that is contained in a function. However, the process in object-oriented systems is distributed over a set of classes. Therefore, testing individual methods makes no sense. The answer is the class. This dramatically changes the manner in which unit testing is done.

A third issue raised is the impact on integration testing. In this case, objects can be aggregated to form aggregate objects; for example, a car has many parts, or they can be grouped together to form collaborations. Furthermore, they can be used in class libraries, frameworks, and components. Based on all of these different ways in which classes can be grouped together, how does one effectively do integration testing?

Polymorphism and Dynamic Binding Polymorphism and dynamic binding dramatically impact both unit and integration testing. Since an individual business process is implemented through a set of methods distributed over a set of objects, as shown above, the unit test makes no sense to be at the method level. However, with polymorphism and dynamic binding, the same method (a small part of the overall business process) can be implemented in many different objects. Therefore, testing individual implementations of methods makes no sense. Again, the unit that makes sense to test is the class. Furthermore, except for trivial cases, dynamic binding makes it impossible to know which implementation is going to be executed until the system does it. Therefore, integration testing becomes very challenging.

Inheritance When considering the issues raised about inheritance (see Chapter 10), it should be no surprise that inheritance impacts the testing of object-oriented systems. Through the use of inheritance, bugs can be propagated from a superclass to all of its direct and indirect subclasses instantaneously. However, the tests that are applicable to a superclass are also applicable to all of its subclasses. As usual, inheritance is a double-edged sword. Finally, even though we have stated this many times before, inheritance should only support a generalization and specialization type of semantics. Remember, when using inheritance the principle of substitutability is critical (see Chapter 7). All of these issues impact unit and integration testing.

Reuse On the surface, reuse should decrease the amount of testing required. However, each time a class is used in a different context, the class must be tested again. Therefore, anytime a class library, framework, or component is used, unit testing and integration testing are important. In the case of a component, the unit to test is the component itself. Remember that a component has a well-defined Application Program Interface (API) that hides the details of its implementation.

Object-Oriented Development Process and Products In virtually all textbooks, including this one, testing is always covered near the end of the SDLC. This seems to imply that testing takes place only after the programming has ended. However, each and every product⁷ that comes out of the object-oriented development process must be tested. For example, it is a lot easier to ensure that the requirements are captured and modeled correctly through testing the use cases. Furthermore, it is a lot cheaper to catch this type of error back in the analysis phase than it is in the implementation phase. Obviously, this is true for testing collaborations as well. By the time we have implemented a collaboration as a set of layers and partitions, we could have expended a great deal of time, and time is money, on implementing the wrong thing. So, testing collaborations by role-playing the CRC cards back in the analysis phase will actually save the team lots of time and money.

⁷ For example, use-case descriptions, use-case diagrams, CRC cards, class diagrams, object diagrams, sequence diagrams, communication diagrams, behavioral state machines, package diagrams, use scenarios, window navigation diagrams, real use cases, contracts, method specifications, and source code.

Testing must take place throughout the SDLC, not simply at the end. However, the type of testing that can take place on nonexecutable representations, such as use cases and CRC cards, is different from those that take place on code written in an object-oriented programming language. The primary approach to testing nonexecutable representations is some form of an inspection or walkthrough of the representation.⁸ Role-playing the CRC cards based on use cases is an example of one type of walkthrough.

Test Planning

Testing starts with the development of a *test plan* that defines a series of *tests* that will be conducted. Since testing takes place through the development of an object-oriented system, a test plan should be developed at the very beginning of the SDLC and continuously updated as the system evolves. The test plan should address all products that are created during the development of the system. For example, tests should be created that can be used to test the completeness of a CRC card. Figure 14-2 shows a typical unit test plan form for a class. For example, Figure 14-3 shows a partial list of the invariant test specifications for a class. Each individual test has a specific objective and describes a set of very specific *test cases* to examine. In the case of invariant-based tests, a description of the invariant, the original values of the attribute, the event that will cause the attribute value to change, the actual results observed, the expected results, and whether it passed or failed are shown. *Test specifications*

Class Test Plan		Page _____ of _____
Class Name: _____		Version Number: _____ CRC Card ID: _____
Tester: _____		Date Designed: _____ Date Conducted: _____
Class Objective:		
Associated Contract IDs: _____		
Associated Use Case IDs: _____		
Associated Superclass(es): _____		
Testing Objectives:		
Walkthrough Test Requirements:		
Invariant-Based Test Requirements:		
State-Based Test Requirements:		
Contract-Based Test Requirements:		

TEMPLATE
can be found at
[www.wiley.com/
college/dennis](http://www.wiley.com/college/dennis)

FIGURE 14-2
Class Test Plan

⁸ See Michael Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, 15, no. 3 (1976): 105–211.

TEMPLATE
can be found at
www.wiley.com/college/dennis

FIGURE 14-3
Class Invariant Test Specification

Class Invariant Test Specification					
Page _____ of _____					
Class Name: _____		Version Number: _____		CRC Card ID: _____	
Tester: _____		Date Designed: _____		Date Conducted: _____	
Testing Objectives:					
Test Cases					
Invariant Description	Original Attribute Value	Event	New Attribute Value	Expected Result	Result P/F
Attribute Name:					
1)	_____	_____	_____	_____	_____
2)	_____	_____	_____	_____	_____
3)	_____	_____	_____	_____	_____
Attribute Name:					
1)	_____	_____	_____	_____	_____
2)	_____	_____	_____	_____	_____
3)	_____	_____	_____	_____	_____
Attribute Name:					
1)	_____	_____	_____	_____	_____
2)	_____	_____	_____	_____	_____
3)	_____	_____	_____	_____	_____

are created for each type of constraint that must be met by the class. Also, similar types of specifications are done for integration, system, and acceptance tests.

Not all classes are likely to be finished at the same time, so the programmer usually writes *stubs* for the unfinished classes to enable the classes around them to be tested. A stub is a placeholder for a class that usually displays a simple test message on the screen or returns some *hardcoded value*⁹ when it is selected. For example, consider an application system that provides the five standard processes discussed in Chapter 6 for some object such as CDs, patients, or employees: creating, changing, deleting, finding, and printing (whether on the screen or on a printer). Depending on the final design, these different processes could end up in different objects on different layers. Therefore, to test the functionality associated with the classes on the problem domain layer, a stub would be written for each of the classes on the other layers that interact with the problem domain classes. These stubs would contain the minimal interface necessary to be able to test the problem domain classes. For example, they would have methods that could receive the messages being sent by the problem domain layer objects and methods that could send messages to the problem domain layer objects. Typically, the methods would display a message on the screen notifying the tester that the method had been successfully reached (e.g., “Delete item from

⁹ The word “hardcoded” means written into the program. For example, suppose you were writing a unit to calculate the net present value of a loan. The stub might be written so that it would always display (or return to the calling module) a value of 100 regardless of the input values.

Database method reached”). In this way, the problem domain classes could pass class testing before the classes on the other layers were completed.

Unit Tests

Unit tests focus on a single unit—the class. There are two approaches to unit testing: black-box and white-box (see Figure 14-4). *Black-box testing* is the most commonly used since each class represents an encapsulated object. Black-box testing is driven by the CRC cards, behavioral state machines, and contracts associated with a class, not by the programmers’ interpretation. In this case, the test plan is developed directly from the specification of the class: each item in the specification becomes a test, and several test cases are developed for it. *White-box testing* is based on the method specifications associated with each class. However, white-box testing has had limited impact in object-oriented development. This is due to the rather small size of the individual methods in a class. As such, most approaches to testing classes use black-box testing to assure their correctness.

Class tests should be based on the invariants on the CRC cards, the behavioral state machines associated with each class, and the pre- and post-conditions contained on each method’s contract. Assuming all of the constraints have been captured on the CRC cards and contracts, individual test cases can be developed fairly easily. For example, suppose the CRC card for an order class gave an invariant that the order quantity must be between 10 and 100 cases. The tester develops a series of test cases to ensure that the quantity is validated before the system accepts it. It is impossible to test every possible combination of input and situation; there are simply too many possible combinations. In this example, the test requires a minimum of three test cases: one with a valid value (e.g., 15), one with an invalid value too low (e.g., 7), and one with invalid value too high (e.g., 110). Most tests would also include a test case with a nonnumeric value to ensure the data types were checked (e.g., ABCD). A really good test would include a test case with nonsensical but potentially valid data (e.g., 21.4).

Using a behavioral state machine is a useful way to identify tests for a class. Any class that has a behavioral state machine associated with it will have a potentially complex life cycle. As such, it is possible to create a series of tests to guarantee that each state can be reached.

Tests also can be developed for each contract associated with the class. In the case of a contract, a set of tests for each pre- and post-condition is required. Furthermore, if the class is a subclass of another class, then all of the tests associated with the superclass must be executed again. Also, the interactions among the constraints, invariants, and pre- and post-conditions in the subclass and the superclass(es) must be addressed.

Finally, owing to good object-oriented design, in order to fully test a class, special testing methods may have to be added to the class being tested. For example, how can invariants be tested? The only way to really test them is through methods that are visible to the outside of the class that can be used to manipulate the values of the class’s attributes. However, adding these types of methods to a class does two things. First, they add to the testing requirements since they themselves will have to be tested. Second, if they are not removed from the deployed version of the system, the system will be less efficient and the advantage of information hiding effectively will be lost. As is readily apparent, testing classes is complex. Therefore, great care must be taken when designing tests for classes.

Integration Tests

Integration tests assess whether a set of classes that must work together do so without error. They ensure that the interfaces and linkages between different parts of the system work properly. At this point, the classes have passed their individual unit tests, so the focus now is on the flow of control among the classes and on the data exchanged among them. Integration testing follows the same general procedures as unit testing: the tester develops a

Stage	Types of Tests	Test Plan Source	When to Use	Notes
Unit Testing	Black-Box Testing Treats class as "black-box"	CRC Cards Class Diagrams Contracts	For normal unit testing	<ul style="list-style-type: none"> Tester focuses on whether the class meets the requirements stated in the specifications.
	White-Box Testing Looks inside the class to test its major elements	Method Specifications	When complexity is high	<ul style="list-style-type: none"> By looking inside the class to review the code itself the tester may discover errors or assumptions not immediately obvious to someone treating the class as a black box.
Integration Testing	User Interface Testing The tester tests each interface function	Interface Design	For normal integration testing	<ul style="list-style-type: none"> Testing is done by moving through each and every menu item in the interface either in a top-down or bottom-up manner.
	Use-Case Testing The tester tests each use case	Use Cases	When the user interface is important	<ul style="list-style-type: none"> Testing is done by moving through each use case to ensure they work correctly. Usually combined with user interface testing because it does not test all interfaces.
	Interaction Testing Tests each process in a step-by-step fashion	Class Diagrams Sequence Diagrams Communication Diagrams	When the system performs data processing	<ul style="list-style-type: none"> The entire system begins as a set of stubs. Each class is added in turn and the results of the class compared to the correct result from the test data; when a class passes, the next class is added and the test rerun. This is done for each package. Once each package has passed all tests, then the process repeats integrating the packages.
	System Interface Testing Tests the exchange of data with other systems	Use-Case Diagram	When the system exchanges data	<ul style="list-style-type: none"> Because data transfers between systems are often automated and not monitored directly by the users it is critical to design tests to ensure they are being done correctly.
System Testing	Requirements Testing Tests to whether original business requirements are met	System Design, Unit Tests, and Integration Tests	For normal system testing	<ul style="list-style-type: none"> Ensures that changes made as a result of integration testing did not create new errors.
	Usability Testing Tests how convenient the system is to use	Interface Design and Use Cases	When user interface is important	<ul style="list-style-type: none"> Often done by analyst with experience in how users think and in good interface design. Sometimes uses formal usability testing procedures discussed in Chapter 12.
	Security Testing Tests disaster recovery & unauthorized access	Infrastructure Design	When the system is important	<ul style="list-style-type: none"> Security testing is a complex task, usually done by an infrastructure analyst assigned to the project. In extreme cases, a professional firm may be hired.
	Performance Testing Examines the ability to perform under high loads	System Proposal Infrastructure Design	When the system is important	<ul style="list-style-type: none"> High volumes of transactions are generated and given to the system. Often done by using special purpose testing software.
	Documentation Testing Tests the accuracy of the documentation	Help System, Procedures, Tutorials	For normal system testing	<ul style="list-style-type: none"> Analysts spot check or check every item on every page in all documentation to ensure the documentation items and examples work properly.
Acceptance Testing	Alpha Testing Conducted by users to ensure they accept the system	System Tests	For normal acceptance testing	<ul style="list-style-type: none"> Often repeats previous tests but are conducted by users themselves to ensure they accept the system.
	Beta Testing Uses real data, not test data	No plan	When the system is important	<ul style="list-style-type: none"> Users closely monitor system for errors or useful improvements.

FIGURE 14-4 Types of Tests

**YOUR
TURN**
14-1 Test Planning for an ATM

Pretend you are a project manager for a bank developing software for ATMs. Develop a unit test plan for the user interface component of the ATM.

test plan that has a series of tests that in turn have tests. Integration testing is often done by a set of programmers and/or systems analysts.

From an object-oriented systems perspective, integration testing can be difficult. A single class can be in many different aggregations, because of the way in which objects can be combined to form new objects, class libraries, frameworks, components, and packages. Where is the best place to start the integration? Typically, the answer is to begin with the set of classes, or collaboration, used to support the highest priority use case (see Chapter 6). Also, dynamic binding makes it crucial to design the integration tests carefully to ensure that the combinations of methods are tested.

There are four approaches to integration testing: *user interface testing*, *use-case testing*, *interaction testing*, and *system interface testing* (see Figure 14-4). Most projects use all four approaches.

System Tests

System tests are usually conducted by the systems analysts to ensure that all classes work together without error. System testing is similar to integration testing but is much broader in scope. Whereas integration testing focuses on whether the classes work together without error, system tests examine how well the system meets business requirements (*requirements testing*), how convenient the system is to use (*usability testing*), how secure the system is (*security testing*), how well the system performs under a heavy load (*performance testing*), and how accurate the documentation of the system is (*documentation testing*). Figure 14-4 provides additional details on the different types of system tests.

CONCEPTS
IN ACTION
14-B Anatomy of a Friendly Hack

If you've seen the movie *Sneakers*, you know that there are professional security firms that organizations can hire to break in to their own networks to test security. BABank (a pseudonym) was about to launch a new online banking application, so it hired such a firm to test its security before the launch. The bank's system failed the security test—badly.

The security team began by mapping the bank's network. It used network security analysis software and dialing software to test for dial-in ports. The mapping process found a bunch of accounts with default passwords (i.e., passwords set by the manufacturer that are supposed to be changed when the systems are first set up). The team

then tricked several high-profile users into revealing their passwords to gain access to several high-privilege accounts. Once into these computers, the team used password cracking software to find passwords on these computers and ultimately gain the administrator passwords on several servers. At this point, the team transferred \$1,000 into their test account. They could have transferred more, but the security point was made.

Source: *Network World*, February 2, 1998.

Question

1. What was the value of the security tests in this case?

Acceptance Tests

Acceptance testing is done primarily by the users with support from the project team. The goal is to confirm that the system is complete, meets the business needs that prompted the system to be developed, and is acceptable to the users. Acceptance testing is done in two stages: *alpha testing*, in which users test the system using made-up data, and *beta testing*, in which users begin to use the system with real data but are carefully monitored for errors (see Figure 14-4).

DEVELOPING DOCUMENTATION

Like testing, developing documentation of the system must be done throughout the SDLC. There are two fundamentally different types of documentation: system documentation and user documentation. *System documentation* is intended to help programmers and systems analysts understand the application software and enable them to build it or maintain it after the system is installed. System documentation is largely a byproduct of the systems analysis and design process and is created as the project unfolds. Each step and phase produces documents that are essential in understanding how the system is constructed or is to be built. In many object-oriented development environments, it is possible to somewhat automate the creation of detailed documentation for classes and methods. For example, in Java, if the programmers use *javadoc*-style comments, it is possible to create HTML pages that document a class and its methods automatically by using the *javadoc* utility.¹⁰ Since most programmers look on documentation with much distaste, anything that can make documentation easier to create is useful.

User documentation (such as user's manuals, training manuals, and online help systems) is designed to help the user operate the system. Although most project teams expect users to have received training and to have read the user's manuals before operating the system, unfortunately this is not always the case. It is more common today—especially in the case of commercial software packages for microcomputers—for users to begin using the software without training or reading the user's manuals. In this section, we focus on user documentation.¹¹

User documentation is often left until the end of the project, which is a dangerous strategy. Developing good documentation takes longer than many people expect because it requires much more than simply writing a few pages. Producing documentation requires designing the documents (whether on paper or online), writing the text, editing them, and testing them. For good-quality documentation, this process usually takes about three hours per page (single-spaced) for paper-based documentation or two hours per screen for online documentation. Thus, a “simple” set of documentation such as a ten-page user's manual and a set of twenty help screens takes seventy hours. Of course, lower quality documentation can be produced faster.

The time required to develop and test user documentation should be built into the project plan. Most organizations plan for documentation development to start once the interface design and program specifications are complete. The initial draft of documentation is usually scheduled for completion immediately after the unit tests are complete. This reduces the chance that the documentation will need to be changed due to software changes (but doesn't eliminate it) and still leaves enough time for the documentation to be tested and revised before the acceptance tests are started.

¹⁰ For those who have used Java, *javadoc* is how the JDK documentation from Sun was created.

¹¹ For more information on developing documentation, see Thomas T. Barker, *Writing Software Documentation* (Boston, MA: Allyn and Bacon, 1998).

Although paper-based manuals are still important, online documentation is becoming more important. Paper-based documentation is simpler to use because it is more familiar to users, especially novices who have less computer experience; online documentation requires the users to learn one more set of commands. Paper-based documentation is also easier to flip through and gain a general understanding of its organization and topics and can be used far away from the computer itself.

There are four key strengths of online documentation that all but guarantee it will be the dominant form for the next century. First, searching for information is often simpler (provided the help search index is well designed) because the user can type in a variety of keywords to view information almost instantaneously rather than having to search through the index or table of contents in a paper document. Second, the same information can be presented several times in many different formats, so that the user can find and read the information in the most informative way (such redundancy is possible in paper documentation, but the cost and intimidating size of the resulting manual make it impractical). Third, online documentation enables many new ways for the user to interact with the documentation that are not possible in static paper documentation. For example, it is possible to use links or “tool tips” (i.e., pop-up text; see Chapter 12) to explain unfamiliar terms, and one can write “show-me” routines that demonstrate on the screen exactly what buttons to click and text to type. Finally, online documentation is significantly less expensive to distribute than paper documentation.

Types of Documentation

There are three fundamentally different types of user documentation: reference documents, procedures manuals, and tutorials. *Reference documents* (also called the help system) are designed to be used when the user needs to learn how to perform a specific function (e.g., updating a field, adding a new record). Often people read reference information when they have tried and failed to perform the function; writing reference documents requires special care because often the user is impatient or frustrated when he or she begins to read them.

Procedures manuals describe how to perform business tasks (e.g., printing a monthly report, taking a customer order). Each item in the procedures manual typically guides the user through a task that requires several functions or steps in the system. Therefore, each entry is typically much longer than an entry in a reference document.

Tutorials—obviously—teach people how to use major components of the system (e.g., an introduction to the basic operations of the system). Each entry in the tutorial is typically longer still than the entries in procedures manuals and is usually designed to be read in sequence (whereas entries in reference documents and procedures manuals are designed to be read individually).

Regardless of the type of user documentation, the overall process for developing it is similar to the process of developing interfaces (see Chapter 12). The developer first designs the general structure for the documentation and then develops the individual components within it.

Designing Documentation Structure

In this section, we focus on the development of online documentation because we believe it will become the most common form of user documentation. The general structure used in most online documentation, whether reference documents, procedures manuals, or tutorials is to develop a set of *documentation navigation controls* that lead the user to *documentation topics*. The documentation topics are the material the user wants to read, while the navigation controls are the way in which the user locates and accesses a specific topic.

Designing the structure of the documentation begins by identifying the different types of topics and navigation controls that need to be included. Figure 14-5 shows a commonly used structure for online reference documents (i.e., the help system). The documentation topics generally come from three sources. The first and most obvious source of topics is the set of commands and menus in the user interface. This set of topics is very useful if the user wants to understand how a particular command or menu is used.

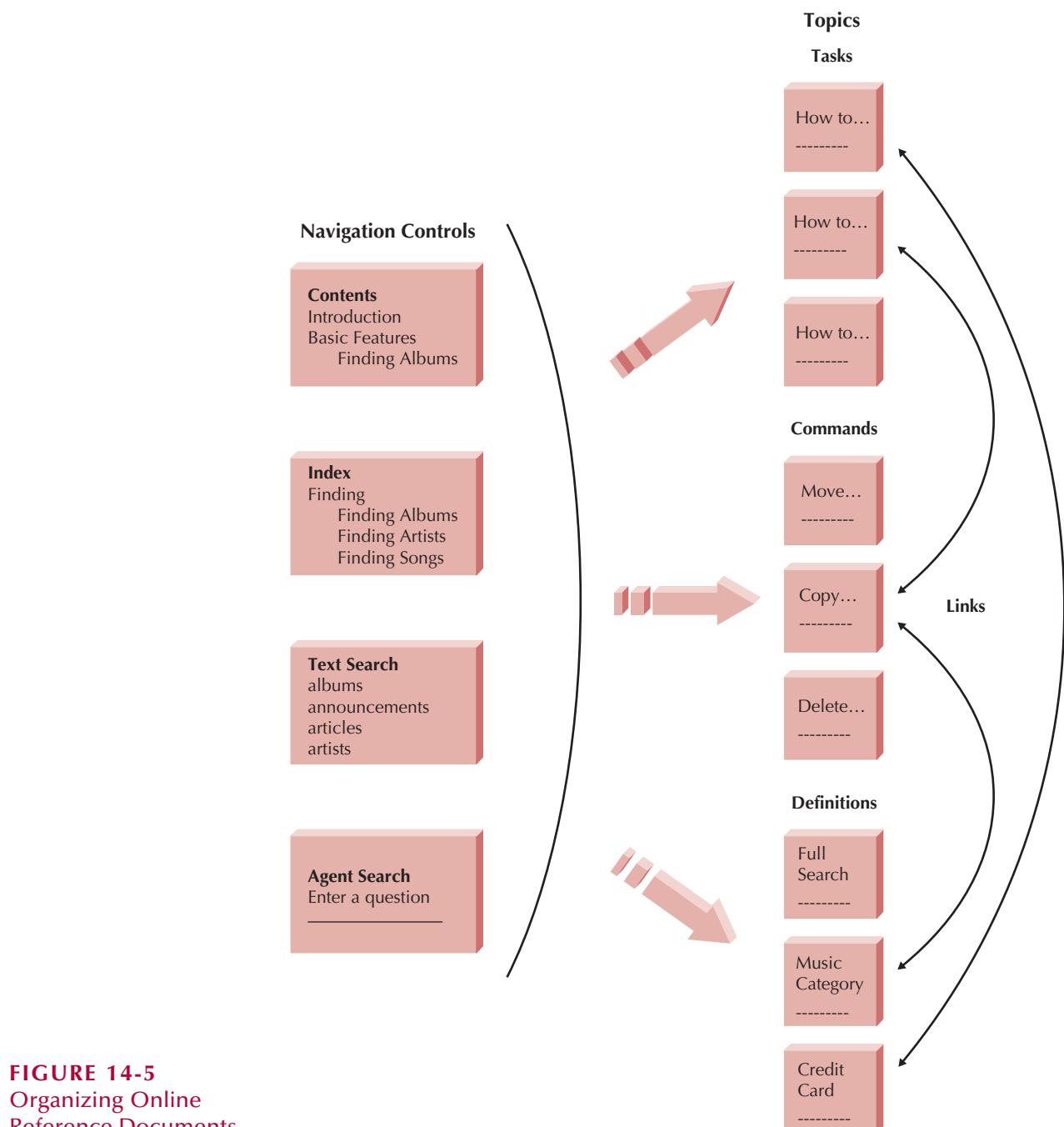


FIGURE 14-5
Organizing Online Reference Documents

The users often don't know what commands to look for or where they are in the system's menu structure. Instead, users have tasks they want to perform, and rather than thinking in terms of commands, they think in terms of their business tasks. Therefore, the second and often more useful set of topics focuses on how to perform certain tasks—usually those in the use scenarios, window navigation diagram, and the real use cases from the user interface design (see Chapter 12). These topics walk the user through the set of steps (often involving several keystrokes or mouse clicks) needed to perform some task.

The third set of topics is definitions of important terms. These terms are usually the use cases and classes in the system, but sometimes they also include commands.

There are five general types of navigation controls for topics, but not all systems use all five types (see Figure 14-5). The first is the table of contents that organizes the information in a logical form, as though the users were to read the reference documentation from start to finish.

The index provides access to the topics based on important keywords, in the same way that the index at the back of a book helps you find topics. Text search provides the ability to search through the topics either for any text the user types or for words that match a developer-specified set of words that is much larger than the words in the index. Unlike the index, text search typically provides no organization to the words (other than alphabetical). Some systems provide the ability to use an intelligent agent to help in the search (e.g., the Microsoft Office Assistant—also known as the paper clip guy). The fifth and final navigation control to topics are the Web-like links between topics that enable the user to click and move among topics.

Procedures manuals and tutorials are similar but often simpler in structure. Topics for procedures manuals usually come from the use scenarios, window navigation diagram, and the real use cases developed during interface design and from other basic tasks the users must perform. Topics for tutorials are usually organized around major sections of the system and the level of experience of the user. Most tutorials start with the basic, most commonly used commands and then move into more complex and less frequently used commands.

Writing Documentation Topics

The general format for topics is fairly similar across application systems and operating systems (see Figure 14-6). Topics typically start with very clear titles, followed by some introductory text that defines the topic, and then provide detailed, step-by-step instructions on how to perform what is being described (where appropriate). Many topics include screen images to help the user find items on the screen; some also have "show-me" examples in which the series of keystrokes and/or mouse movements and clicks needed to perform the function are demonstrated to the user. Most also include navigation controls to enable movement among topics, usually at the top of the window, plus links to other topics. Some also include links to "related topics" that include options or other commands and tasks the user may want to perform in concert with the topic being read.

Writing the topic content can be challenging. It requires a good understanding of the user (or more accurately the range of users) and a knowledge of what skills the user(s) currently have and can be expected to import from other systems and tools they are using or have used (including the system the new system is replacing). Topics should always be written from the viewpoint of the user and describe what the user wants to accomplish, not what the system can do. Figure 14-7 provides some general guidelines to improve the quality of documentation text.¹²

¹² One of the best books to explain the art of writing is William Strunk and E.B. White, *Elements of Style*, 4th ed. (Boston: Allyn and Bacon, 2000).

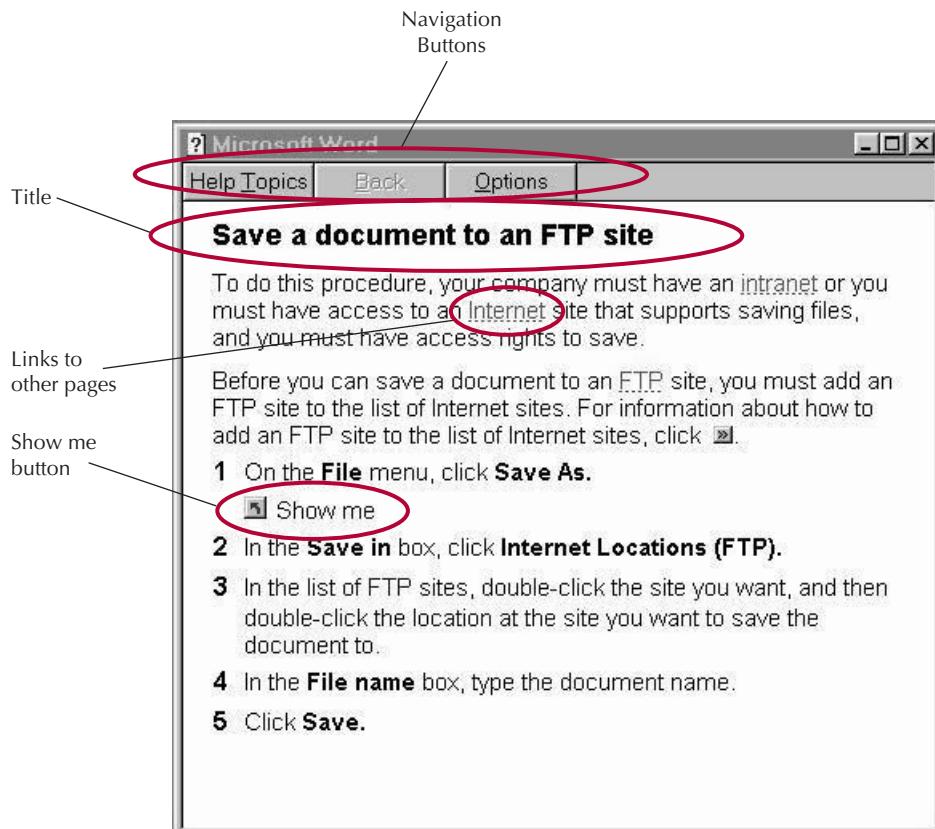


FIGURE 14-6
A Help Topic in
Microsoft Word

Identifying Navigation Terms

As you write the documentation topics you also begin to identify the terms that will be used to help users find topics. The table of contents is usually the most straightforward because it is developed from the logical structure of the documentation topics, whether reference topics, procedure topics, or tutorial topics. The items for the index and search engine require more care because they are developed from the major parts of the system and the users' business functions. Every time you write a topic, you must also list the terms that will be used to find the topic. Terms for the index and search engine can come from four distinct sources.

The first source is the set of the commands in the user interface, such as open file, modify customer, and print open orders. All commands contain two parts (action and object). It is important to develop the index for both parts because users could search for information using either part. A user looking for more information about saving files, for example, might search by using the term "save" or the term "files."

The second source is the set of major concepts in the system, which are often use cases and classes. In the case of CD Selections, for example, this might include music category, album, and shipping costs.

A third source is the set of business tasks the user performs, such as ordering replacement unit or making an appointment. Often these will be contained in the command set, but sometimes they require several commands and use terms that always appear in the system. Good sources for these terms are the use scenarios and real use cases developed during interface design (see Chapter 12).

A fourth, often controversial, source is the set of synonyms for the three sets of items above. Users sometimes don't think in terms of the nicely defined terms used by the system. They may try to find information on how to "stop" or "quit" rather than "exit," or "erase" rather than "delete." Including synonyms in the index increases the complexity and size of the documentation system but can greatly improve the usefulness of the system to the users.

Guideline	Before the Guideline	After the Guideline
Use the active voice: The active voice creates more active and readable text by putting the subject at the start of the sentence, the verb in the middle, and the object at the end.	Finding albums is done using the album title, the artist's name, or a song title.	You can find an album by using the album title, the artist's name, or a song title.
Use e-prime style: E-prime style creates more active writing by omitting all forms of the verb to be.	The text you want to copy must be selected before you click on the copy button.	Select the text you want to copy before you click on the copy button.
Use consistent terms: Always use the same term to refer to the same items, rather than switching among synonyms (e.g., change, modify, update).	Select the text you want to copy. Pressing the copy button will copy the marked text to the new location.	Select the text you want to copy. Pressing the copy button will copy the selected text to the new location.
Use simple language: Always use the simplest language possible to accurately convey the meaning. This does not mean you should "dumb down" the text but that you should avoid artificially inflating its complexity. Avoid separating subjects and verbs and try to use the fewest words possible. (When you encounter a complex piece of text, try eliminating words; you may be surprised at how few words are really needed to convey meaning.)	The Georgia Statewide Academic and Medical System (GSAMS) is a cooperative and collaborative distance learning network in the state of Georgia. The organization in Atlanta that administers and manages the technical and overall operations of the currently more than 300 interactive audio and video teleconferencing classrooms throughout Georgia system is the Department of Administrative Service (DOAS). (56 words)	The Department of Administrative Service (DOAS) in Atlanta manages the Georgia Statewide Academic and Medical System (GSAMS), a distance learning network with more than 300 teleconferencing classrooms throughout Georgia. (29 words)
Use friendly language: Too often, documentation is cold and sterile because it is written in a very formal manner. Remember, you are writing for a person, not a computer.	Blank disks have been provided to you by Operations. It is suggested that you ensure your data is not lost by making backup copies of all essential data.	You should make a backup copy of all data that is important to you. If you need more diskettes, contact Operations.
Use parallel grammatical structures: Parallel grammatical structures indicate the similarity among items in list and help the reader understand content.	Opening files Saving a document How to delete files	Opening a file Saving a file Deleting a file
Use steps correctly: Novices often interperse action and the results of action when describing a step-by-step process. Steps are always actions.	1. Press the customer button. 2. The customer dialogue box will appear. 3. Type the customer ID and press the submit button and the customer record will appear.	1. Press the customer button. 2. Type the customer ID in the customer dialogue box when it appears. 3. Press the submit button to view the customer record for this customer.
Use short paragraphs: Readers of documentation usually quickly scan text to find the information they need, so the text in the middle of long paragraphs is often overlooked. Use separate paragraphs to help readers find information more quickly.		

Source: Adapted from T. T. Barker, *Writing Software Documentation* (Boston: Allyn & Bacon, 1998).

FIGURE 14-7 Guidelines for Crafting Documentation Topics

**YOUR
TURN**
14-2 Documentation for an ATM

Pretend you are a project manager for a bank developing software for ATMs. Develop the part of an online help system that deals with withdrawing and depositing cash. Be

sure to include the different types of accounts, such as checking and savings.

APPLYING THE CONCEPTS AT CD SELECTIONS

Managing Programming

Three programmers were assigned by CD Selections to develop the three major parts of the Internet sales system. The first was the Web interface, both the client side (browser) and the server side. The second was the client–server-based management system (managing the CD information and marketing materials databases). The third was the interfaces between the Internet sales system and CD Selections' existing distribution system and the credit card center. Programming went smoothly and despite a few minor problems proceeded according to plan.

Testing

While the programmers were working, Alec began developing the test plans and user documentation. The test plans for the three components were similar but slightly more intensive for the Web interface component (see Figure 14-8). Unit testing would use black-box testing based on the CRC cards, class diagrams, and contracts for all components. Figure 14-9 shows part of the class-invariant test specification for the Order class in the Web interface component.

Integration testing for the Web interface and system management component would be subjected to all user interface and use-case tests to ensure that the interface would work properly. The system interface component would undergo system interface tests to ensure that the system performed calculations properly and was capable of exchanging data with the CD Selections' other systems and the credit card center.

Systems tests are by definition tests of the entire system—all components together. However, not all parts of the system would receive the same level of testing. Requirements tests would be conducted on all parts of the system to ensure that all requirements were met. Because security was a critical issue, the security of all aspects of the system would be tested. Security tests would be developed by CD Selections' infrastructure team, and once the system passed those tests, an external security consulting firm would be hired to attempt to break in to the system.

Performance was an important issue for the parts of the system used by the customer (the Web interface and the system interfaces to the credit card and inventory systems) but not as important for the management component that would be used by staff,

Test Stage	Web Interface	System Management	System Interfaces
Unit tests	Black-box tests	Black-box tests	Black-box tests
Integration tests	User interface tests; use-case tests	User interface tests; use-case tests	System interface tests
System tests	Requirements tests; security tests; performance tests; usability tests	Requirements tests; security tests	Requirements tests; security tests; performance tests
Acceptance tests	Alpha test; beta test	Alpha test; beta test	Alpha test; beta test

FIGURE 14-8
CD Selections'
Test Plan

Class Invariant Test Specification					
Page <u>5</u> of <u>15</u>					
Class Name: <u>Order</u>		Version Number: <u>3</u>		CRC Card ID: <u>15</u>	
Tester: <u>Susan Doe</u>		Date Designed: <u>9/9</u>		Date Conducted: _____	
Testing Objectives: Ensure that the information entered by the customer on the place order form is valid.					
Test Cases					
Invariant Description	Original Attribute Value	Event	New Attribute Value	Expected Result	Result P/F
Attribute Name: CD Number					
1) <u>(1..1)</u>	<u>Null</u>	CreateOrder	<u>Null</u>	<u>F</u>	_____
2) <u>(unsigned long)</u>	<u>Null</u>	CreateOrder	<u>ABC</u>	<u>F</u>	_____
3) <u>(unsigned long)</u>	<u>Null</u>	CreateOrder	<u>123</u>	<u>P</u>	_____
Attribute Name: CD Name					
1) <u>(1..1)</u>	<u>Null</u>	CreateOrder	<u>Null</u>	<u>F</u>	_____
2) <u>(String)</u>	<u>Null</u>	CreateOrder	<u>ABC</u>	<u>P</u>	_____
3) <u>(String)</u>	<u>Null</u>	CreateOrder	<u>123</u>	<u>P</u>	_____
Attribute Name: Artist Name					
1) <u>(1..1)</u>	<u>Null</u>	CreateOrder	<u>Null</u>	<u>F</u>	_____
2) <u>(String)</u>	<u>Null</u>	CreateOrder	<u>ABC</u>	<u>P</u>	_____
3) <u>(String)</u>	<u>Null</u>	CreateOrder	<u>123</u>	<u>P</u>	_____

FIGURE 14-9
Partial Class Invariant Test Specification for the Order Class

not customers. The customer-facing components would undergo rigorous performance testing to see how many transactions (whether searching or purchasing) they could handle before they were unable to provide a response time of two seconds or less. Alec also developed an upgrade plan so that as demand on the system increased, there was a clear plan for when and how to increase the processing capability of the system.

Finally, formal usability tests would be conducted on the Web interface portion of the system with six potential users (both novice and expert Internet users).

Acceptance tests would be conducted in two stages, alpha and beta. Alpha tests would be done during the training of CD Selections' staff. The Internet Sales Group manager would work together with Alec to develop a series of tests and training exercises to train the Internet Sales Group staff in how to use the system. They would then load the real CD data into the system and begin adding marketing materials. These same staff and other CD Selections staff members would also pretend to be customers and test the Web interface.

Beta testing would be done by “going live” with the Web site but its existence would only be announced to CD Selections’ employees. As an incentive to try the Web site (rather than buying from the store in which they worked), employees would be offered triple their normal employee discount for all products ordered from the Web site. The site would also have a prominent button on every screen that would enable employees to e-mail comments to the project team, and the announcement would encourage employees to report problems, suggestions, and compliments to the project team. After one month, assuming all went well, the beta test would be completed, and the Internet sales site would be linked to the main Web site and advertised to the general public.

Developing User Documentation

Three types of documentation (reference documents, procedures manuals, and tutorials) could be produced for the Web interface and the management component. Since the number of CD Selections' staff using the system management component would be small, Alec decided to produce only the reference documentation (an online help system). He felt that an intensive training program and a one-month beta test period would be sufficient without tutorials and formal procedures manuals. Similarly, he felt that the process of ordering CDs and the Web interface itself was simple enough to not require a tutorial on the Web—a help system would be sufficient (and a procedures manual didn't make sense).

Alec decided that the reference documents for both the Web interface and system management components would contain help topics for user tasks, commands, and definitions. He also decided that the documentation component would contain four types of navigation controls: a table of contents, an index, a find, and links to definitions. He did not feel that the system was complex enough to benefit from a search agent.

After these decisions were made, Alec delegated the development of the reference documents to a technical writer assigned to the project team. Figure 14-10 shows examples of a few of topics that the writer developed. The tasks and commands were taken directly from the interface design. The list of definitions was developed once the tasks and commands were developed based on the writer's experience in understanding what terms might be confusing to the user.

Once the topic list was developed, the technical writer then began writing the topics themselves as well as the navigation controls to access. Figure 14-11 shows an example of one topic taken from the task list: how to place an order. This topic presents a brief description of what it is and then leads the user through the step-by-step process needed to complete the task. The topic also lists the navigation controls that will be used to find the topic in terms of the table of contents entries, the index entries, and search entries. It also lists what words in the topic itself will have links to other topics (e.g., shopping cart).

SUMMARY

Managing Programming

Programming is done by programmers, so systems analysts have few responsibilities during this stage. The project manager, however, is usually very busy. The first task is to assign the programmers, ideally the fewest possible to complete the project because coordination problems increase as the size of the programming team increases. Coordination can be improved by having regular meetings, ensuring that standards are followed, implementing change control, and using CASE tools effectively. One of the key functions of the project manager is to manage the schedule and adjust it for delays. Two common causes of delays are scope creep and minor slippages that go unnoticed.

Tasks	Commands	Terms
Find an album	Find	Album
Add an album to my shopping cart	Browse	Artist
Placing an order	Quick search	Music type
How to buy	Full search	Special deals
What's in my shopping cart?		Cart
		Shopping cart

FIGURE 14-10
Sample Help Topics for
CD Selections

FIGURE 14-11
Example Documentation Topic for CD Selections

The screenshot shows a help topic titled "How to Place an Order". The main content area contains the following text:

When you are ready to pay for the merchandise you have selected (the items in your shopping cart) you can place your order. There are four steps.

1. Move to the Place order Page

Click on the **Place order** button to move to the place order page.

2. Make sure you are ordering what you want

The place order screen displays all the items in your shopping cart. Read through the list to make sure these are what you want because once you submit your credit card information you cannot change the order.

you can delete an item by

On the right side, there is a sidebar titled "Navigation Controls" with the following sections:

- Table of Contents** list: How to Place an Order
- Index** list: Credit Card, Order, Pay, Place order
- Search** find by: Credit Card, Delete Items, Order, Pay, Place order, Shopping Cart, Verify Order
- Links:** Shopping Cart

Test Planning

Tests must be carefully planned because the cost of fixing one major bug after the system is installed can easily exceed the annual salary of a programmer. A test plan contains several tests that examine different aspects of the system. A test, in turn, specifies several test cases that will be examined by the testers. A unit test examines a class within the system; test cases come from the class specifications or the class code itself. An integration test examines how well several classes work together; test cases come from the interface design, use cases, the use case diagram, sequence diagrams, and communication diagrams. A system test examines the system as a whole and is broader than the unit and integration tests; test cases come from the system design, the infrastructure design, and the unit and integration tests. Acceptance testing is done by the users to determine whether the system is acceptable to them; it draws on the system test plans (alpha testing) and on the real work the users perform (beta testing).

Documentation

Documentation, both user documentation and system documentation, is moving away from paper-based documents to online documentation. There are three types of user documentation: reference documents are designed to be used when the user needs to learn how to perform a specific function (e.g., an online help system), procedures manuals describe how to perform business tasks, while tutorials teach people how to use the system. Documentation navigation controls (e.g., a table of contents, index, find, intelligent agents, or links between pages) enable users to find documentation topics (e.g., how to perform a function, how to use an interface command, an explanation of a term).

KEY TERMS

Acceptance test
Alpha test
Beta test
Black-box testing

Change control
Construction
Documentation navigation control
Documentation testing

Documentation topic
Hardcoded value
Integration test
Interaction testing

Performance testing	Slippage	Test specification
Procedures manual	Stub	Tutorial
Program log	System documentation	Unit test
Reference document	System interface testing	Usability testing
Requirements testing	System test	Use-case testing
Risk assessment	Test	User documentation
Scope creep	Test case	User interface testing
Security testing	Test plan	White-box testing

QUESTIONS

1. Why is testing important?
2. What is the primary role of systems analysts during the programming stage?
3. In *The Mythical Man-Month*, Frederick Brooks argues that adding more programmers to a late project makes it later. Why?
4. What is the purpose of testing?
5. Describe how object-orientation impacts testing.
6. Compare and contrast the terms test, test plan, and test case.
7. What is a stub, and why is it used in testing?
8. What is the primary goal of unit testing?
9. Compare and contrast black-box testing and white-box testing.
10. How are the test cases developed for unit tests?
11. What are the different types of unit tests?
12. What is the primary goal of integration testing?
13. How are the test cases developed for integration tests?
14. What is the primary goal of system testing?
15. How are the test cases developed for system tests?
16. What is the primary goal of acceptance testing?
17. How are the test cases developed for acceptance tests?
18. Compare and contrast alpha testing and beta testing.
19. Compare and contrast user documentation and system documentation.
20. Why is online documentation becoming more important?
21. What are the primary disadvantages of online documentation?
22. Compare and contrast reference documents, procedures manuals, and tutorials.
23. What are five types of documentation navigation controls?
24. What are the commonly used sources of documentation topics? Which is the most important? Why?
25. What are the commonly used sources of documentation navigation controls? Which is the most important? Why?

EXERCISES

- A. Create an invariant test specification for the class you chose for the Health Club in Exercise A in Chapter 10.
- B. Create an invariant test specification for the class you chose for the Picnics R Us in Exercise C in Chapter 10.
- C. Create an invariant test specification for the class you chose for the Of-The-Month Club (OTMC) in Exercise E in Chapter 10.
- D. Create an invariant test specification for the class you chose for the Library in Exercise G in Chapter 10.
- E. If the registration system at your university does not have a good online help system, develop an online help system for one screen of the user interface.
- F. Examine and prepare a report on the online help system for the calculator program in Windows (or a similar on the Mac or Unix). (You will likely be surprised at the amount of help for such a simple program.)
- G. Compare and contrast the online help at two different Web sites that enable you to perform some function (e.g., make travel reservations, order books).

MINICASES

1. Pete is a project manager on a new systems development project. This project is Pete's first experience as a project manager, and he has led his team successfully to the programming phase of the project. The project

has not always gone smoothly, and Pete has made a few mistakes, but he is generally pleased with the progress of his team and the quality of the system being developed. Now that programming has begun,

Pete has been hoping for a little break in the hectic pace of his workday.

Prior to beginning programming, Pete recognized that the time estimates made earlier in the project were too optimistic. However, he was firmly committed to meeting the project deadline because of his desire for his first project as project manager to be a success. In anticipation of this time pressure problem, Pete arranged with the Human Resources department to bring in two new college graduates and two college interns to beef up the programming staff. Pete would have liked to find some staff with more experience, but the budget was too tight, and he was committed to keeping the project budget under control.

Pete made his programming assignments, and work on the programs began about two weeks ago. Now, Pete has started to hear some rumbles from the programming team leaders that may signal trouble. It seems that the programmers have reported several instances where they wrote programs, only to be unable to find them when they went to test them. Also, several programmers have opened programs that they had written, only to find that someone had changed portions of their programs without their knowledge.

- a. Is the programming phase of a project a time for the project manager to relax? Why or why not?
 - b. What problems can you identify in this situation? What advice do you have for the project manager? How likely does it seem that Pete will achieve his desired goals of being on time and within budget if nothing is done?
2. The systems analysts are developing the test plan for the user interface for the Holiday Travel Vehicles system. As the salespeople are entering a sales invoice into the system, they will be able to either enter an option code into a text box, or select an option code from a drop-down list. A combo box was used to implement this, since it was felt that the salespeople would quickly become familiar with the most common option codes and would prefer entering them directly to speed up the entry process.

It is now time to develop the test for validating the option code field during data entry. If the customer did not request any dealer-installed options for the vehicle, the salesperson should enter “none”; the field should not be blank. The valid option codes are four-character alphabetic codes and should be matched against a list of valid codes.

Prepare a test plan for the test of the option code field during data entry.

CHAPTER 15

INSTALLATION AND OPERATIONS

This chapter examines the activities needed to install the information system and successfully convert the organization to using it. It also discusses post-implementation activities, such as system support, system maintenance, and project assessment. Installing the system and making it available for use from a technical perspective is relatively straightforward. However, the training and organizational issues surrounding the installation are more complex and challenging because they focus on people, not computers.

OBJECTIVES

- Be familiar with the system installation process.
- Understand different types of conversion strategies and when to use them.
- Understand several techniques for managing change.
- Be familiar with post-installation processes.

CHAPTER OUTLINE

Introduction	Motivating Adoption
Conversion	Enabling Adoption: Training
Conversion Style	Post-Implementation Activities
Conversion Location	System Support
Conversion Modules	System Maintenance
Selecting the Appropriate Conversion	Project Assessment
Strategy	Applying the Concepts at CD Selections
Change Management	Conversion
Understanding Resistance to Change	Change Management
Revising Management Policies	Post-Implementation Activities
Assessing Costs and Benefits	Summary

INTRODUCTION

It must be remembered that there is nothing more difficult to plan, more doubtful of success, nor more dangerous to manage than the creation of a new system. For the initiator has the animosity of all who would profit by the preservation of the old institution and merely lukewarm defenders in those who would gain by the new.

—Machiavelli, *The Prince*, 1513

Although written almost 500 years ago, Machiavelli's comments are still true today. Managing the change to a new system—whether it is computerized or not—is one of the most difficult tasks in any organization. Because of the challenges involved, most organizations begin developing their conversion and change management plans while the programmers are still developing the software. Leaving conversion and change management planning to the last minute is a recipe for failure.

In many ways, using a computer system or set of work processes is much like driving on a dirt road. Over time with repeated use, the road begins to develop ruts in the most commonly used parts of the road. Although these ruts show where to drive, they make change difficult. As people use a computer system or set of work processes, those systems/work processes begin to become habits or norms; people learn them and become comfortable with them. These systems or work processes then begin to limit people's activities and make it difficult for them to change because they begin to see their jobs in terms of these processes rather than of the final business goal of serving customers.

One of the earliest models for managing organizational change was developed by Kurt Lewin.¹ Lewin argued that change is a three-step process: unfreeze, move, refreeze (Figure 15-1). First, the project team must *unfreeze* the existing habits and norms (the As-Is system) so that change is possible. Most of the systems development life cycle (SDLC) to this point has laid the groundwork for unfreezing. Users are aware of the new system being developed, some have participated in an analysis of the current system (and so are aware of its problems), and some have helped design the new system (and so have some sense of the potential benefits of the new system). These activities have helped to unfreeze the current habits and norms.

The second step is to help the organization move to the new system via a migration plan. The migration plan has two major elements. One is technical, which includes how the new system will be installed and how data in the As-Is system will be moved into the To-Be system; this is discussed in the Conversion section of this chapter. The second component is organizational, which includes helping users understand the change and motivating them to adopt it; this is discussed in the Change Management section of this chapter.

The third step is to *refreeze* the new system as the habitual way of performing the work processes—ensuring that the new system successfully becomes the standard way of performing the business function it supports. This refreezing process is a key goal of the Post-implementation Activities discussed in the final section of this chapter. By providing

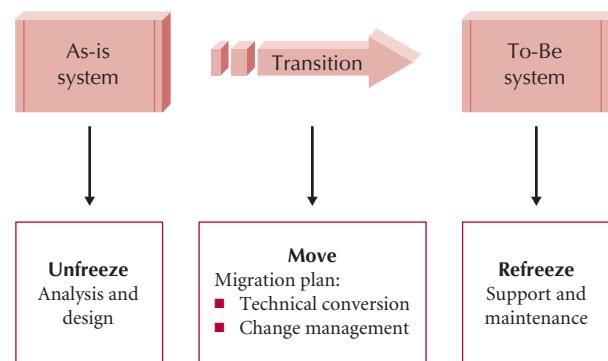


FIGURE 15-1
Implementing Change

¹ Kurt Lewin, "Frontiers in Group Dynamics," *Human Relations*, 1, no. 5 (1947:5–41), and Kurt Lewin, "Group Decision and Social Change" in E.E. Maccoby, T.M. Newcomb, and E.L. Hartley (eds.), *Readings in Social Psychology* (New York: Holt, Rinehart & Winston, 1958), pp. 197–211.

ongoing support for the new system and immediately beginning to identify improvements for the next version of the system, the organization helps solidify the new system as the new habitual way of doing business. Post-implementation activities include system support, which means providing help desk and telephone support for users with problems; system maintenance, which means fixing bugs and improving the system after it has been installed; and project assessment, which means the process of evaluating the project to identify what went well and what could be improved for the next system development project.

Change management is the most challenging of the three components because it focuses on people, not technology, and because it is the one aspect of the project that is the least controllable by the project team. Change management means winning the hearts and minds of potential users and convincing them that the new system actually provides value.

Maintenance is the most costly aspect of the installation process, because the cost of maintaining systems usually greatly exceeds the initial development costs. It is not unusual for organizations to spend 60 to 80 percent of their total IS development budget on maintenance. Although this may sound surprising initially, think about the software you use. How many software packages do you use that are the very first version? Most commercial software packages become truly useful and enter widespread use only in their second or third version. Maintenance and continual improvement of software is ongoing, whether it is a commercially available package or software developed in-house. Would you buy software if you knew that no new versions were going to be produced? Of course, commercial software is somewhat different from custom in-house software used by only one company, but the fundamental issues remain.

Project assessment is probably the least commonly performed part of the SDLC but is perhaps the one that has the most long-term value to the IS department. Project assessment enables project team members to step back and consider what they did right and what they could have done better. It is an important component in the individual growth and development of each member of the team, because it encourages team members to learn from their successes and failures. It also enables new ideas or new approaches to system development to be recognized, examined, and shared with other project teams to improve their performance.

CONVERSION

Conversion is the technical process by which the new system replaces the old system. Users are moved from using the As-Is business processes and computer programs to the To-Be business processes and programs. The *migration plan* specifies what activities will be performed when and by whom and includes both technical aspects (such as installing hardware and software and converting data from the As-Is system to the To-Be system) and organizational aspects (such as training and motivating the users to embrace the new system). Conversion refers to the technical aspects of the migration plan.

There are three major steps to the conversion plan before commencement of operations: install hardware, install software, and convert data (Figure 15-2). Although it may be possible to do some of these steps in parallel, usually they must be done sequentially at any one location.

The first step in the conversion plan is to buy and install any needed hardware. In many cases, no new hardware is needed, but sometimes the project requires such new hardware as servers, client computers, printers, and networking equipment. It is critical to work closely with vendors who are supplying needed hardware and software to ensure that the deliveries are coordinated with the conversion schedule so that the equipment is available when it is needed. Nothing can stop a conversion plan in its tracks as easily as the failure of a vendor to deliver needed equipment.

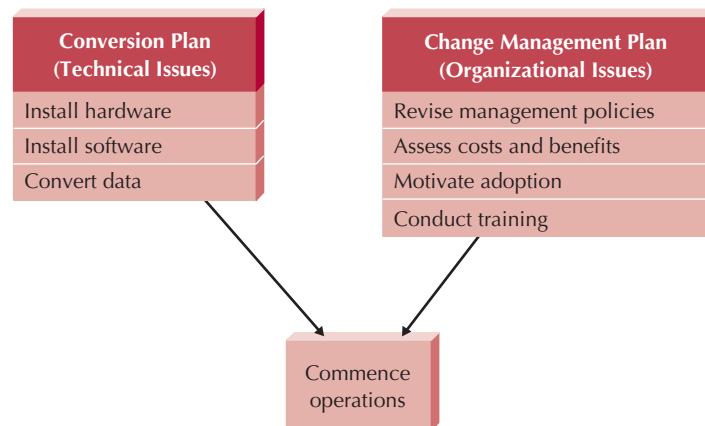


FIGURE 15-2
Elements of a
Migration Plan

Once the hardware is installed, tested, and certified as being operational, the second step is to install the software. This includes the To-Be system under development, and sometimes additional software that must be installed to make the system operational. For example, the CD Selections Internet sales system needs Web server software. At this point, the system is usually tested again to ensure that it operates as planned.

The third step is to convert the data from the As-Is system to the To-Be system. Data conversion is usually the most technically complicated step in the migration plan. Often, separate programs must be written to convert the data from the As-Is system to the new formats required in the To-Be system and store it in the To-Be system files and databases. This process is often complicated by the fact that the files and databases in the To-Be system do not exactly match the files and databases in the As-Is system (e.g., the To-Be system may use several tables in a database to store customer data that was contained in one file in the As-Is system). Formal test plans are always required for data conversion efforts (see Chapter 14).

Conversion can be thought of along three dimensions: the style by which the conversion is done (*conversion style*), what location or work groups are converted at what time (*conversion location*), and what modules of the system are converted at what time (*conversion modules*). Figure 15-3 shows the potential relationships among these three dimensions.

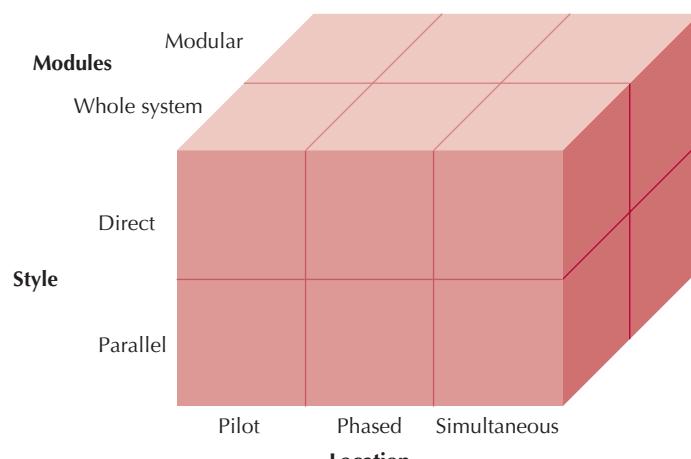


FIGURE 15-3
Conversion Strategies

Conversion Style

The conversion style is the way in which users are switched between the old and new systems. There are two fundamentally different approaches to the style of conversion: direct conversion and parallel conversion.

Direct Conversion With *direct conversion* (sometimes called cold turkey, big bang, or abrupt cutover), the new system instantly replaces the old system. The new system is turned on and the old system is immediately turned off. This is the approach that you probably use when you upgrade commercial software (e.g., Microsoft Word) from one version to another; you simply begin using the new version and stop using the old version.

Direct conversion is the simplest and most straightforward. However, it is the most risky, because any problems with the new system that have escaped detection during testing may seriously disrupt the organization.

Parallel Conversion With *parallel conversion*, the new system is operated side by side with the old system; both systems are used simultaneously. For example, if a new accounting system is installed, the organization enters data into both the old system and the new system and then carefully compares the output from both systems to ensure that the new system is performing correctly. After some time period (often one to two months) of parallel operation and intense comparison between the two systems, the old system is turned off and the organization continues using the new system.

This approach is more likely to catch any major bugs in the new system and prevent the organization from suffering major problems. If problems are discovered in the new system, the system is simply turned off and fixed and then the conversion process starts again. The problem with this approach is the added expense of operating two systems that perform the same function.

Conversion Location

Conversion location refers to those parts of the organization that are converted at what points in time. Often, parts of the organization are physically located in different offices (e.g., Toronto, Atlanta, Los Angeles). In other cases, location refers to different organizational units located in different parts of the same office complex (e.g., order entry, shipping, purchasing). There are at least three fundamentally different approaches to selecting the way in which different organizational locations are converted: pilot conversion, phased conversion, and simultaneous conversion.

Pilot Conversion With a *pilot conversion*, one or more locations or units/work groups within a location are selected to be converted first as part of a pilot test. The locations participating in the pilot test are converted (using either direct or parallel conversion). If the system passes the pilot test, then the system is installed at the remaining locations (again using either direct or parallel conversion).

Pilot conversion has the advantage of providing an additional level of testing before the system is widely deployed throughout the organization, so that any problems with the system affect only the pilot locations. However, this type of conversion obviously requires more time before the system is installed at all organizational locations. Also, it means that different organizational units are using different versions of the system and business processes, which may make it difficult for them to exchange data.

Phased Conversion With *phased conversion*, the system is installed sequentially at different locations. A first set of locations are converted, then a second set, then a third set,

CONCEPTS

IN ACTION

15-A Converting to the Euro (Part 1)

When the European Union decided to introduce the euro, the European Central Bank had to develop a new computer system (called Target) to provide a currency settlement system for use by investment banks and brokerages. Prior to the introduction of the euro, settlement was performed between the central banks of the countries involved. After the introduction, the Target system, which consists of fifteen national banking systems, would settle trades and perform currency conversions for cross-border payments for stocks and bonds.

Source: Thomas Hoffman, "Debut of Euro Nearly Flawless," *Computerworld*, 33, no. 2 (January 11, 1999): 16.

Question:

1. Implementing Target was a major undertaking for a number of reasons. If you were an analyst on the project, what kinds of issues would you have to address to make sure the conversion happened successfully?

and so on, until all locations are converted. Sometimes there is a deliberate delay between the different sets (at least between the first and the second), so that any problems with the system are detected before too much of the organization is affected. In other cases, the sets are converted back-to-back so that as soon as those converting one location have finished, the project team moves to the next and continues the conversion.

Phased conversion has the same advantages and disadvantages of pilot conversion. In addition, it means that a smaller set of people are required to perform the actual conversion (and any associated user training) than if all locations were converted at once.

Simultaneous Conversion *Simultaneous conversion*, as the name suggests, means that all locations are converted at the same time. The new system is installed and made ready at all locations, and at a preset time, all users begin using the new system. Simultaneous conversion is often used with direct conversion, but it can also be used with parallel conversion.

Simultaneous conversion eliminates problems with having different organizational units using different systems and processes. However, it also means that the organization must have sufficient staff to perform the conversion and train the users at all locations simultaneously.

Conversion Modules

Although it is natural to assume that systems are usually installed in their entirety, this is not always the case.

Whole System Conversion *Whole-system conversions*, in which the entire system is installed at one time, is the most common. It is simple and the easiest to understand. However, if the system is large and/or extremely complex (e.g., an enterprise resource planning system such as SAP or PeopleSoft), the whole system may prove too difficult for users to learn in one conversion step.

Modular Conversion When the *modules*² within a system are separate and distinct, organizations sometimes choose to convert to the new system one module at a time—that is, *modular conversion*. Modular conversion requires special care in developing the system

² In this case, a module is typically a component or a package, that is, a set of collaborating classes.

(and usually adds extra cost). Each module either must be written to work with both the old and new systems or object wrappers (see Chapter 9) must be used to encapsulate the old system from the new. When modules are tightly integrated, this is very challenging and therefore seldom done. However, when there is only a loose association between modules, this becomes easier. For example, consider a conversion from an old version of Microsoft Office to a new version. It is relatively simple to convert from the old version of Word to the new version without simultaneously having to change from the old to the new version of Microsoft Excel.

Modular conversion reduces the amount of training required to begin using the new system. Users need training in only the new module being implemented. However, modular conversion does take longer and has more steps than does the whole-system process.

Selecting the Appropriate Conversion Strategy

Each of the three dimensions in Figure 15-3 is independent, so that a conversion strategy can be developed to fit in any one of the boxes in this figure. Different boxes can also be mixed and matched into one *conversion strategy*. For example, one commonly used approach is to begin with a pilot conversion of the whole system using parallel conversion in a handful of test locations. Once the system has passed the pilot test at these locations, it is then installed in the remaining locations using phased conversion with direct cutover. There are three important factors to consider in selecting a conversion strategy: *risk*, *cost*, and the *time* required (Figure 15-4).

Risk After the system has passed a rigorous battery of unit, system, integration, and acceptance testing, it should be bug free ... maybe. Because humans make mistakes, nothing built by people is ever perfect. Even after all these tests, there may still be a few undiscovered bugs. The conversion process provides one last step in which to catch these bugs before the system goes live and the bugs have the chance to cause problems.

Parallel conversion is less risky than is direct conversion because it has a greater chance of detecting bugs that have gone undiscovered in testing. Likewise, pilot conversion is less risky than is phased conversion or simultaneous conversion because if bugs do occur, they occur in pilot test locations whose staff are aware that they may encounter bugs. Since potential bugs affect fewer users, there is less risk. Likewise, converting a few modules at a time lowers the probability of a bug because there is more likely to be a bug in the whole system than in any given module.

How important the risk is depends on the system being implemented—the combination of the probability that bugs remain undetected in the system and the potential cost of those undetected bugs. If the system has indeed been subjected to extensive methodical testing, including alpha and beta testing, then the probability of undetected bugs is lower than if the testing was less rigorous. However, there still might have been mistakes made in

Characteristic	Conversion Style			Conversion Location			Conversion Modules	
	Direct Conversion	Parallel Conversion	Pilot Conversion	Phased Conversion	Simultaneous Conversion	Whole-System Conversion	Modular Conversion	
Risk	High	Low	Low	Medium	Medium	High	High	Medium
Cost	Low	High	Medium	Medium	High	Medium	High	High
Time	Short	Long	Medium	Long	Short	Short	Long	Long

FIGURE 15-4 Characteristics of Conversion Strategies

**YOUR
TURN**

15-1 Developing a Conversion Plan

Suppose you are leading the conversion from one word processor to another at your university. Develop a conversion plan (i.e., technical issues only). You have also been asked to develop a conversion plan for

the university's new Web-based course registration system. How would the second conversion plan be similar to and different from the one you developed for the word processor?

the analysis process, so that although there might be no software bugs, the software might fail to properly address the business needs.

Assessing the cost of a bug is challenging, but most analysts and senior managers can make a reasonable guess at the relative cost of a bug. For example, the cost of a bug in an automated stock market trading program or a heart-lung machine keeping someone alive is likely to be much greater than a bug in a computer game or word processing program. Therefore, risk is likely to be a very important factor in the conversion process if the system has not been as thoroughly tested as it might have been and/or if the cost of bugs is high. If the system has been thoroughly tested and/or the cost of bugs is not that high, then risk becomes less important to the conversion decision.

Cost As might be expected, different conversion strategies have different costs. These costs can include things such as salaries for people who work with the system (e.g., users, trainers, system administrators, external consultants), travel expenses, operation expenses, communication costs, and hardware leases. Parallel conversion is more expensive than direct cutover because it requires that two systems (the old and the new) be operated at the same time. Employees must now perform twice the usual work because they have to enter the same data into both the old and the new systems. Parallel conversion also requires the results of the two systems to be completely cross-checked to make sure there are no differences between the two, which entails additional time and cost.

Pilot conversion and phased conversion have somewhat similar costs. Simultaneous conversion has higher costs because more staff are required to support all the locations as they simultaneously switch from the old to the new system. Modular conversion is more expensive than whole system conversion because it requires more programming. The old system must be updated to work with selected modules in the new system, and modules in the new system must be programmed to work with selected modules in both the old and new systems.

Time The final factor is the amount of time required to convert between the old and the new system. Direct conversion is the fastest because it is immediate. Parallel conversion takes longer because the full advantages of the new system do not become available until the old system is turned off. Simultaneous conversion is fastest because all locations are converted at the same time. Phased conversion usually takes longer than pilot conversion because usually (but not always) once the pilot test is complete all remaining locations are simultaneously converted. Phased conversion proceeds in waves, often requiring several months before all locations are converted. Likewise, modular conversion takes longer than whole system conversion because the models are introduced one after another.

CONCEPTS**IN ACTION****15-B U.S. Army Installation Support**

Throughout the 1960s, 1970s, and 1980s, the U.S. Army automated its installations (army bases, in civilian terms). Automation was usually a local effort at each of the more than 100 bases. Although some bases had developed software together (or borrowed software developed at other bases), each base often had software that performed different functions or performed the same function in different ways. In 1989, the army decided to standardize the software so that the same software would be used everywhere. This would greatly reduce software maintenance and also reduce training when soldiers were transferred between bases.

The software took four years to develop. The system was quite complex and the project manager was concerned that there was a high risk that not all requirements of all installations had been properly captured. Cost and time were less important, since the project had already run four years and cost \$100 million.

Therefore, the project manager chose a modular pilot conversion using parallel conversion. The manager selected seven installations, each representing a different type of army installation (e.g., training base, arsenal, depot) and began the conversion. All went well, but several new features were identified that had been overlooked during the analysis, design, and construction. These were added and the pilot testing resumed. Finally, the system was installed in the rest of the army installations using a phased direct conversion of the whole system.

—Alan Dennis

Question:

1. Do you think the conversion strategy was appropriate?
2. Regardless of whether you agree, what other conversion strategy could have been used?

CHANGE MANAGEMENT

In the context of a systems development project, *change management* is the process of helping people to adopt and adapt to the To-Be system and its accompanying work processes without undue stress.³ There are three key roles in any major organizational change. The first is the *sponsor* of the change—the person who wants the change. This person is the business sponsor who first initiated the request for the new system (see Chapter 3). Usually, the sponsor is a senior manager of the part of the organization that must adopt and use the new system. It is critical that the sponsor be active in the change management process because a change that is clearly being driven by the sponsor, not by the project team or the IS organization, has greater legitimacy. The sponsor has direct management authority over those who adopt the system.

The second role is that of the *change agent*—the person(s) leading the change effort. The change agent, charged with actually planning and implementing the change, is usually someone outside of the business unit adopting the system and therefore has no direct management authority over the potential adopters. Because the change agent is an outsider from a different organizational culture, he or she has less credibility than do the sponsor and other members of the business unit. After all, once the system has been installed, the change agent usually leaves and thus has no ongoing impact.

The third role is that of *potential adopter* or target of the change—the people who actually must change. These are the people for whom the new system is designed and who will ultimately choose to use or not use the system.

³ Many books have been written on change management. Some of our favorites are the following: Patrick Connor and Linda Lake, *Managing Organizational Change*, 2nd ed. (Westport, CT: Praeger, 1994); Douglas Smith, *Taking Charge of Change* (Reading, MA: Addison-Wesley, 1996); and Daryl Conner, *Managing at the Speed of Change* (New York: Villard Books, 1992).

In the early days of computing, many project teams simply assumed that their job ended when the old system was converted to the new system at a technical level. The philosophy was “build it and they will come.” Unfortunately, that happens only in the movies. Resistance to change is common in most organizations. Therefore, the change management plan is an important part of the overall installation plan that glues together the key steps in the change management process. Successful change requires that people want to adopt the change and are able to adopt the change. The change management plan has four basic steps: revising management policies, assessing the cost and benefit models of potential adopters, motivating adoption, and enabling people to adopt through training (see Figure 15-2). However, before we can discuss the change management plan, we must first understand why people resist change.

Understanding Resistance to Change⁴

People resist change—even change for the better—for very rational reasons. What is good for the organization is not necessarily good for the people who work there. For example, consider an order-processing clerk who used to receive orders to be shipped on paper shipping documents but now uses a computer to receive the same information. Rather than typing shipping labels with a typewriter, the clerk now clicks on the print button on the computer and the label is produced automatically. The clerk can now ship many more orders each day, which is a clear benefit to the organization. The clerk, however, probably doesn’t really care how many packages are shipped. His or her pay doesn’t change; it’s just a question of which the clerk prefers to use, a computer or typewriter. Learning to use the new system and work processes—even if the change is minor—requires more effort than continuing to use the existing well-understood, system and work processes.

So why do people accept change? Simply put, every change has a set of costs and benefits associated with it. If the benefits of accepting the change outweigh the costs of the change, then people change. And sometimes the benefit of change is avoidance of the pain that you would experience if you did not adopt the change (e.g., if you don’t change, you are fired, so one of the benefits of adopting the change is that you still have a job).

In general, when people are presented with an opportunity for change, they perform a cost–benefit analysis (sometime consciously, sometimes subconsciously) and decide the extent to which they will embrace and adopt the change. They identify the costs of and benefits from the system and decide whether the change is worthwhile. However, it is not that simple, because most costs and benefits are not certain. There is some uncertainty as to whether a certain benefit or cost will actually occur; so both the costs of and benefits from the new system will need to be weighted by the degree of certainty associated with them (Figure 15-5). Unfortunately, most humans tend to overestimate the probability of costs and underestimate the probability of benefits.

There are also costs and sometimes benefits associated with the actual *transition process* itself. For example, suppose you found a nicer house or apartment than your current one. Even if you liked it better, you might decide not to move simply because the cost of moving outweighed the benefits from the new house or apartment itself. Likewise, adopting a new computer system might require you to learn new skills, which could be seen as a cost to some people, or as a benefit to others, if they perceived that those skills would somehow provide other benefits beyond the use of the system itself. Once again, any costs and benefits from the transition process must be weighted by the certainty with which they will occur (see Figure 15-5).

⁴ This section benefited from conversations with Dr. Robert Briggs, research scientist at the Center for the Management of Information at the University of Arizona.

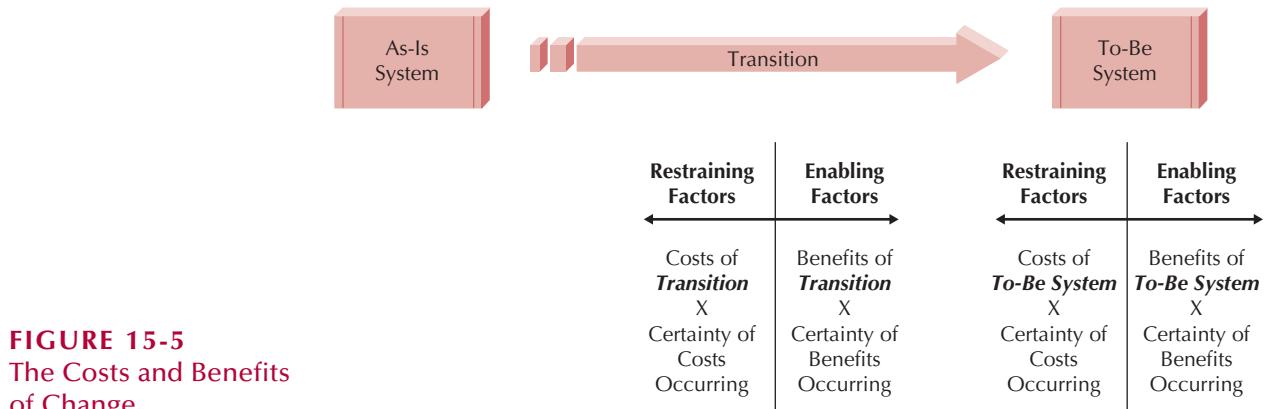


FIGURE 15-5
The Costs and Benefits
of Change

Taken together, these two sets of costs and benefits (and their relative certainties) affect the acceptance of change or resistance to change that project teams encounter when installing new systems in organizations. The first step in change management is to understand the factors that inhibit change—the factors that affect the *perception* of costs and benefits and certainty that they will be generated by the new system. It is critical to understand that the *real costs* and *real benefits* are far less important than the *perceived costs* and *perceived benefits*. People act on what they *believe* to be true, not on what *is* true. Thus, any understanding of how to motivate change must be developed from the viewpoint of the people expected to change, not from the viewpoint of those leading the change.

Revising Management Policies

The first major step in the change management plan is to change the management policies that were designed for the As-Is system to new management policies designed to support the To-Be system. *Management policies* provide goals, define how work processes should be performed, and determine how organizational members are rewarded. No computer system will be successfully adopted unless management policies support its adoption. Many new computer systems bring changes to business processes; they enable new ways of working. Unless the policies that provide the rules and rewards for those processes are revised to reflect the new opportunities that the system permits, potential adopters cannot easily use it.

Management has three basic tools for structuring work processes in organizations.⁵ The first are the *standard operating procedures (SOPs)* that become the habitual routines for how work is performed. The SOPs are both formal and informal. Formal SOPs define proper behavior. Informal SOPs are the norms that have developed over time for how processes are actually performed. Management must ensure that the formal SOPs are revised to match the To-Be system. The informal SOPs will then evolve to refine and fill in details absent in the formal SOPs.

The second aspect of management policy is defining how people assign meaning to events. What does it mean to “be successful” or “do good work”? Policies help people understand meaning by defining *measurements* and *rewards*. Measurements explicitly define meaning because they provide clear and concrete evidence about what is important to the organization. Rewards reinforce measurements because “what gets measured gets done” (an

⁵ This section builds on the work of Anthony Giddens, *The Constitution of Society: Outline of the Theory of Structure* (Berkeley: University of California Press, 1984). A good summary of Giddens's theory that has been revised and adapted for use in understanding information systems is an article by Wanda Orlikowski and Dan Robey: “Information Technology and the Structuring of Organizations,” *Information Systems Research*, 2, no. 2 (1991): 143–169.

**YOUR
TURN**

15-2 Standard Operating Procedures

Identify and explain three standard operating procedures whether they are formal or informal. for the course in which you are using this book. Discuss

overused but accurate saying). Measurements must be carefully designed to motivate desired behavior. The IBM credit example (Your Turn 5-2 in Chapter 5) illustrates the problem with flawed measurements' driving improper behavior (when the credit analysts became too busy to handle credit requests, they would find nonexistent errors so they could return them unprocessed).

A third aspect of management policy is *resource allocation*. Managers can have clear and immediate impacts on behavior by allocating resources. They can redirect funds and staff from one project to another, create an infrastructure that supports the new system, and invest in training programs. Each of these activities has both a direct and symbolic effect. The direct effect comes from the actual reallocation of resources. The symbolic effect shows that management is serious about its intentions. There is less uncertainty about management's long-term commitment to a new system when potential adopters see resources being committed to support it.

Assessing Costs and Benefits

The next step in developing a change management plan is to develop two clear and concise lists of costs and benefits provided by the new system (and the transition to it) compared with the As-Is system. The first list is developed from the perspective of the organization, which should flow easily from the business case developed during the feasibility study and refined over the life of the project (see Chapter 3). This set of organizational costs and benefits should be distributed widely so that everyone expected to adopt the new system should clearly understand why the new system is valuable to the organization.

The second list of costs and benefits is developed from the viewpoints of the different potential adopters expected to change, or stakeholders in the change. For example, one set of potential adopters may be the frontline employees, another may be the first-line supervisors, and yet another might be middle management. Each of these potential adopters/stakeholders may have a different set of costs and benefits associated with the change—costs and benefits that can differ widely from those of the organization. In some situations, unions may be key stakeholders that can make or break successful change.

Many systems analysts naturally assume that frontline employees are the ones whose set of costs and benefits are the most likely to diverge from those of the organization and thus are the ones who most resist change. However, they usually bear the brunt of problems with the current system. When problems occur, they often experience them first-hand. Middle managers and first-line supervisors are the most likely to have a divergent set of costs and benefits and therefore resist change because new computer systems often change how much power they have. For example, a new computer system may improve the organization's control over a work process (a benefit to the organization) but reduce the decision-making power of middle management (a clear cost to middle managers).

An analysis of the costs and benefits for each set of potential adopters/stakeholders will help pinpoint those who will likely support the change and those who may resist the change.

The challenge at this point is to try to change the balance of the costs and benefits for those expected to resist the change so that they support it (or at least do not actively resist it).

This analysis may uncover some serious problems that have the potential to block the successful adoption of the system. It may be necessary to reexamine the management policies and make significant changes to ensure that the balance of costs and benefits is such that important potential adopters are motivated to adopt the system.

Figure 15-6 summarizes some of the factors that are important to successful change. The first and most important reason is a compelling personal reason to change. All change is made by individuals, not organizations. If there are compelling reasons for the key groups of individual stakeholders to want the change, then the change is more likely to be successful. Factors such as increased salary, reduced unpleasantness, and—depending on the individuals—opportunities for promotion and personal development can be important motivators. However, if the change makes current skills less valuable, individuals may resist the change because they have invested a lot of time and energy in acquiring those skills and anything that diminishes those skills may be perceived as diminishing the individual (because important skills bring respect and power).

There must also be a compelling reason for the organization to need the change; otherwise, individuals become skeptical that the change is important and are less certain it will in fact occur. Probably the hardest organization to change is an organization that has been successful because individuals come to believe that what worked in the past will continue to work. By contrast, in an organization that is on the brink of bankruptcy, it is easier to convince individuals that change is needed. Commitment and support from credible business sponsors and top management are also important in increasing the certainty that the change will occur.

The likelihood of successful change is increased when the cost of the transition to individuals who must change is low. The need for significantly different new skills or disruptions in operations and work habits may create resistance. A clear migration plan developed by a credible change agent who has support from the business sponsor are important factors in increasing the certainty about the costs of the transition process.

CONCEPTS

15-C Understanding Resistance to a Decision Support System

IN ACTION

One of the first commercial software packages I developed was a decision support system to help schedule orders in a paper mill (see Concepts in Action 5-B in Chapter 5). The system was designed to help the person who scheduled orders decide when to schedule particular orders to reduce waste in the mill. This was a very challenging problem—so challenging, in fact, that it usually took the scheduler a year or two to really learn how to do the job well.

The software was tested by a variety of paper mills over the years and always reduced the amount of waste, usually by about 25 percent but sometimes by 75 percent when a scheduler new to the job was doing the scheduling. Although we ended up selling the package to most paper mills that tested it, we usually encountered significant resistance from the person doing the scheduling

(except when the scheduler was new to the job and the package clearly saved a significant amount). At the time, I assumed that the resistance to the system was related to the amount of waste reduced: the less waste reduced, the more resistance because the payback analysis showed it took longer to pay for the software.

—Alan Dennis

Question:

1. What is another possible explanation for the different levels of resistance encountered at different mills?
2. How might this be addressed?

Motivating Adoption

The single most important factor in motivating a change is providing clear and convincing evidence of the need for change. Simply put, everyone who is expected to adopt the change must be convinced that the benefits from the To-Be system outweigh the costs of changing.

Factor	Examples	Effects	Actions to Take
Benefits of To-Be system	Compelling personal reason(s) for change	Increased pay, fewer unpleasant aspects, opportunity for promotion, most existing skills remain valuable.	If the new system provides clear personal benefits to those who must adopt it, they are more likely to embrace the change. Perform a cost-benefit analysis from the viewpoint of the stakeholders, make changes where needed, and actively promote the benefits.
Certainty of benefits	Compelling organizational reason(s) for change	Risk of bankruptcy, acquisition, government regulation.	If adopters do not understand why the organization is implementing the change, they are less certain that the change will occur. Perform a cost-benefit analysis from the viewpoint of the organization and launch a vigorous information campaign to explain the results to everyone.
	Demonstrated top management support	Active involvement, frequent mentions in speeches.	If top management is not seen to actively support the change, there is less certainty that the change will occur. Encourage top management to participate in the information campaign.
	Committed and involved business sponsor	Active involvement, frequent visits to users and project team, championing.	If the business sponsor (the functional manager who initiated the project) is not seen to actively support the change, there is less certainty that the change will occur. Encourage the business sponsor to participate in the information campaign and play an active role in the change management plan.
	Credible top management and business sponsor	Management and sponsor who do what they say instead of being members of the "management fad of the month" club.	If the business sponsor and top management have credibility in the eyes of the adopters, the certainty of the claimed benefits is higher. Ensure that the business sponsor and/or top management has credibility so that such involvement will help; if there is no credibility involvement will have little effect.
Costs of transition	Low personal costs of change	Few new skills needed.	The cost of the change is not borne equally by all stakeholders; the costs are likely to be higher for some. Perform a cost-benefit analysis from the viewpoint of the stakeholders, make changes where needed, and actively promote the low costs.
Certainty of costs	Clear plan for change	Clear dates and instructions for change, clear expectations.	If there is a clear migration plan, it will likely lower the perceived costs of transition. Publicize the migration plan.
	Credible change agent	Previous experience with change, does what he/she promises to do.	If the change agent has credibility in the eyes of the adopters, the certainty of the claimed costs is higher. If the change agent is not credible, then change will be difficult.
	Clear mandate for change agent from sponsor	Open support for change agent when disagreements occur.	If the change agent has a clear mandate from the business sponsor, the certainty of the claimed costs is higher. The business sponsor must actively demonstrate support for the change agent.

FIGURE 15-6 Major Factors in Successful Change

There are two basic strategies to motivating adoption: informational and political. Both strategies are often used simultaneously. With an *informational strategy*, the goal is to convince potential adopters that the change is for the better. This strategy works when the cost–benefit set of the target adopters has more benefits than costs. In other words, there really are clear reasons for the potential adopters to welcome the change.

Using this approach, the project team provides clear and convincing evidence of the costs and benefits of moving to the To-Be system. The project team writes memos and develops presentations that outline the costs and benefits of adopting the system from the perspective of the organization and from the perspective of the target group of potential adopters. This information is disseminated widely throughout the target group, much like an advertising or public relations campaign. It must emphasize the benefits as well as increase the certainty in the minds of potential adopters that these benefits will actually be achieved. In our experience, it is always easier to sell painkillers than vitamins; that is, it is easier to convince potential adopters that a new system will remove a major problem (or other source of pain) than that it will provide new benefits (e.g., increase sales). Therefore, informational campaigns are more likely to be successful if they stress the reduction or elimination of problems, rather than focusing on the provision of new opportunities.

The other strategy to motivate change is a *political strategy*. With a political strategy, organizational power, not information, is used to motivate change. This approach is often used when the cost–benefit set of the target adopters has more costs than benefits. In other words, although the change may benefit the organization, there are no reasons for the potential adopters to welcome the change.

The political strategy is usually beyond the control of the project team. It requires someone in the organization who holds legitimate power over the target group to influence the group to adopt the change. This may be done in a coercive manner (e.g., “adopt the system or you’re fired”) or in a negotiated manner, in which the target group gains benefits in other ways that are linked to the adoption of the system (e.g., linking system adoption to increased training opportunities). Management policies can play a key role in a political strategy by linking salary to certain behaviors desired with the new system.

In general, for any change that has true organizational benefits, about 20 percent to 30 percent of potential adopters will be *ready adopters*. They recognize the benefits, quickly adopt the system, and become proponents of the system. Another 20 percent to 30 percent are *resistant adopters*. They simply refuse to accept the change and they fight against it, either because the new system has more costs than benefits for them personally or because they place such a high cost on the transition process itself that no amount of benefits from the new system can outweigh the change costs. The remaining 40 percent to 60 percent are *reluctant adopters*. They tend to be apathetic and will go with the flow to either support or resist the system, depending on how the project evolves and how their coworkers react to the system. Figure 15-7 illustrates the actors who are involved in the change management process.

The goal of change management is to actively support and encourage the ready adopters and help them win over the reluctant adopters. There is usually little that can be

CONCEPTS

15-D Overcoming Resistance to a Decision Support System

IN ACTION

How would you motivate adoption if you were the developer of the decision support system described in Concepts in Action 15-C earlier in this chapter?

FIGURE 15-7
Actors in the Change Management Process

Sponsor	Change Agent	Potential Adopters
The sponsor wants the change to occur.	The change agent leads the change effort.	Potential adopters are the people who must change. 20–30 percent are ready adopters. 20–30 percent are resistant adopters. 40–60 percent are reluctant adopters.

done about the resistant adopters because their set of costs and benefits may be divergent from those of the organization. Unless there are simple steps that can be taken to rebalance their costs and benefits or the organization chooses to adopt a strongly political strategy, it is often best to ignore this small minority of resistant adopters and focus on the larger majority of ready and reluctant adopters.

Enabling Adoption: Training

Potential adopters may want to adopt the change, but unless they are capable of adopting it, they won't. Adoption is enabled by providing the skills needed to adopt the change through careful *training*. Training is probably the most self-evident part of any change management initiative. How can an organization expect its staff members to adopt a new system if they are not trained? However, we have found that training is one of the most commonly overlooked parts of the process. Many organizations and project managers simply expect potential adopters to find the system easy to learn. Since the system is presumed to be so simple, it is taken for granted that potential adopters should be able to learn with little effort. Unfortunately, this is usually an overly optimistic assumption.

Every new system requires new skills either because the basic work processes have changed (sometimes radically in the case of business process reengineering [BPR]; see Chapter 5) or because the computer system used to support the processes is different. The more radical the changes to the business processes, the more important it is to ensure the organization has the new skills required to operate the new business processes and supporting information systems. In general, there are three ways to get these new skills. One is to hire new employees who have the needed skills that the existing staff does not. Another is to outsource the processes to an organization that has the skills that the existing staff does not. Both of these approaches are controversial and are usually considered only in the case of BPR when the new skills needed are likely to be the most different from the set of skills of the current staff. In most cases, organizations choose the third alternative: training existing staff in the new business processes and the To-Be system. Every training plan must consider what to train and how to deliver the training.

What to Train What training should you provide to the system users? It's obvious: how to use the system. The training should cover all the capabilities of the new system so users understand what each module does, right?

Wrong. Training for business systems should focus on helping the users to accomplish their jobs, not on how to use the system. The system is simply a means to an end, not the end in itself. This focus on performing the job (i.e., the business processes), not using the system, has two important implications. First, the training must focus on those activities around the system, as well as on the system itself. The training must help the users understand how the computer fits into the bigger picture of their jobs. The use of the system must be put in context of the manual business processes as well as of those that are

computerized, and it must also cover the new management policies that were implemented along with the new computer system.

Second, the training should focus on what the user needs to do, not what the system can do. This is a subtle—but very important—distinction. Most systems will provide far more capabilities than the users will need to use (e.g., when was the last time you wrote a macro in Microsoft Word?). Rather than attempting to teach the users all the features of the system, training should instead focus on the much smaller set of activities that users perform on a regular basis and ensure that users are truly expert in those. When the focus is on the 20 percent of functions that the users will use 80 percent of the time (instead of attempting to cover all functions), users become confident about their ability to use the system. Training should mention the other little-used functions, but only so that users are aware of their existence and know how to learn about them when their use becomes necessary.

One source of guidance for designing training materials is the use cases. The use cases outline the common activities that users perform and thus can be helpful in understanding the business processes and system functions that are likely to be most important to the users.

How to Train There are many ways to deliver training. The most commonly used approach is *classroom training* in which many users are trained at the same time by the same instructor. This has the advantage of training many users at one time with only one instructor and creates a shared experience among the users.

It is also possible to provide *one-on-one training* in which one trainer works closely with one user at a time. This is obviously more expensive, but the trainer can design the training program to meet the needs of individual users and can better ensure that the users really do understand the material. This approach is typically used only when the users are very important or when there are very few users.

Another approach that is becoming more common is to use some form of *computer-based training (CBT)*, in which the training program is delivered via computer, either on CD or over the Web. CBT programs can include text slides, audio, and even video and animation. CBT is typically more costly to develop but is cheaper to deliver because no instructor is needed to actually provide the training.

Figure 15-8 summarizes four important factors to consider in selecting a training method: cost to develop, cost to deliver, impact, and reach. CBT is typically more expensive to develop than one-on-one or classroom training, but it is less expensive to deliver. One-on-one training has the most impact on the user because it can be customized to the user's precise needs, knowledge, and abilities, whereas CBT has the least impact. However, CBT has the greatest reach—the ability to train the most users over the widest distance in the shortest time—because it is much simpler to distribute, than classroom and one-on-one training, because no instructors are needed.

Figure 15-8 suggests a clear pattern for most organizations. If there are only a few users to train, one-on-one training is the most effective. If there are many users to train, many organizations turn to CBT. We believe that the use of CBT will increase in the future. Quite

YOUR TURN

15-3 Developing a Training Plan

Suppose you are leading the conversion from one word processor to another in your organization. Develop an

outline of topics that would be included in the training. Develop a plan for training delivery.

FIGURE 15-8
Selecting a Training Method

	One-on-One Training	Classroom Training	Computer-Based Training
Cost to develop	Low–Medium	Medium	High
Cost to deliver	High	Medium	Low
Impact	High	Medium–High	Low–Medium
Reach	Low	Medium	High

often, large organizations use a combination of all three methods. Regardless of which approach is used, it is important to leave the users with a set of easily accessible materials that can be referred to long after the training has ended (usually a quick reference guide and a set of manuals, whether on paper or in electronic form).

POST-IMPLEMENTATION ACTIVITIES

The goal of post-implementation activities is the *institutionalization* of the use of the new system—that is to make it the normal, accepted, routine way of performing the business processes. The *post-implementation* activities attempt to refreeze the organization after the successful transition to the new system. Although the work of the project team naturally winds down after implementation, the business sponsor and sometimes the project manager are actively involved in refreezing. These two—and ideally many other stakeholders—actively promote the new system and monitor its adoption and usage. They usually provide a steady flow of information about the system and encourage users to contact them to discuss issues.

In this section, we examine three key post-implementation activities: *system support* (providing assistance in the use of the system), *system maintenance* (continuing to refine and improve the system), and *project assessment* (analyzing the project to understand what activities were done well—and should be repeated—and what activities need improvement in future projects).

System Support

Once the project team has installed the system and performed the change management activities, the system is officially turned over to the *operations group*. This group is responsible for the operation of the system, whereas the project team was responsible for development of the system. Members of the operations group usually are closely involved in the installation activities because they are the ones who must ensure that the system actually works. After the system is installed, the project team leaves but the operations group remains.

Providing system support means helping the users to use the system. Usually, this means providing answers to questions and helping users understand how to perform a certain function; this type of support can be thought of as *on-demand training*.

Online support is the most common form of on-demand training. This includes the documentation and help screens built into the system, as well as separate Web sites that provide answers to *frequently asked questions (FAQs)* that enable users to find answers without contacting a person. Obviously, the goal of most systems is to provide sufficiently good online support so that the user doesn't need to contact a person, because providing online support is much less expensive than is providing a person to answer questions.

Most organizations provide a *help desk* that provides a place for a user to talk with a person who can answer questions (usually over the phone, but sometimes in person). The help desk supports all systems, not just one specific system, so it receives calls about a wide variety of software and hardware. The help desk is operated by *level 1 support* staff who

have very broad computer skills and are able to respond to a wide range of requests, from network problems and hardware problems to problems with commercial software and problems with the business application software developed in-house.

The goal of most help desks is to have the level 1 support staff resolve 80 percent of the help requests they receive on the first call. If the issue cannot be resolved by level 1 support staff, a *problem report* (Figure 15-9) is completed (often using a special computer system designed to track problem reports) and passed to a *level 2 support* staff member.

The level 2 support staff members are people who know the application system well and can provide expert advice. For a new system, they are usually selected during the implementation phase and become familiar with the system as it is being tested. Sometimes, the level 2 support staff members participate in training during the change management process to become more knowledgeable with the system, the new business processes, and the users themselves.

The level 2 support staff works with users to resolve problems. Most problems are successfully resolved by the level 2 staff. However, sometimes, particularly in the first few months after the system is installed, the problem turns out to be a bug in the software that must be fixed. In this case, the problem report becomes a *change request* that is passed to the system maintenance group (see the next section).

TEMPLATE
can be found at
[www.wiley.com/
college/dennis](http://www.wiley.com/college/dennis)

FIGURE 15-9
Elements of a
Problem Report

- Time and date of the report
- Name, e-mail address, and telephone number of the support person taking the report
- Name, e-mail address, and telephone number of the person who reported the problem
- Software and/or hardware causing problem
- Location of the problem
- Description of the problem
- Action taken
- Disposition (problem fixed or forwarded to system maintenance)

CONCEPTS

15-E Converting to the Euro (Part 2)

IN ACTION

When the European Union decided to introduce the euro, the European Central Bank had to develop a new computer system (called Target) to provide a currency settlement system for use by investment banks and brokerages. The euro opened at an exchange rate of U.S. \$1.167. However, a rumor that the Target system malfunctioned sent the value of the euro plunging two days later.

That evening, it was determined that the malfunction was not due to system problems. Instead, operators at some German banks had misunderstood how to use the system and had entered incorrect data. Once the

problems were identified and the operators quickly retrained, the Target system continued to operate and the euro quickly regained its lost value.

Source: Thomas Hoffman, "Debut of euro nearly flawless," *Computerworld*, 33(2) (January 11, 1999), p. 16.

Question:

1. Target could be considered a high risk system because of its effects on the European economy. What kinds of system support activities could be put in place to mitigate problems with Target?

System Maintenance

System maintenance is the process of refining the system to make sure it continues to meet business needs. Substantially more money and effort is devoted to system maintenance than to the initial development of the system, simply because a system continues to change and evolve as it is used. Most beginning systems analysts and programmers work first on maintenance projects; usually only after they have gained some experience are they assigned to new development projects.

Every system is “owned” by a project manager in the IS group (Figure 15-10). This individual is responsible for coordinating the systems maintenance effort for that system. Whenever a potential change to the system is identified, a change request is prepared and forwarded to the project manager. The change request is a “smaller” version of the *system request* discussed in Chapters 1 and 3. It describes the change requested and explains why the change is important.

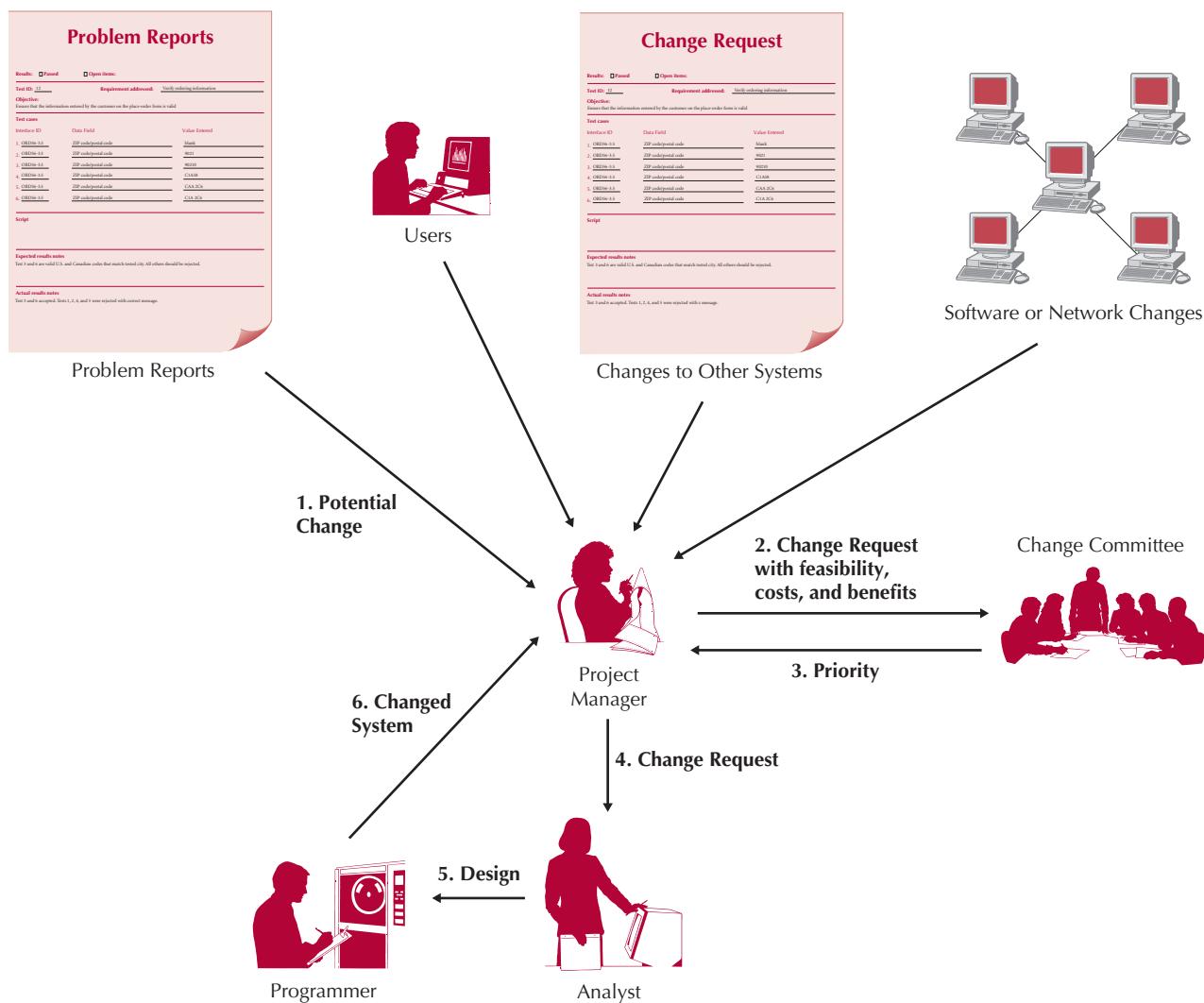


FIGURE 15-10 Processing a Change Request

Changes can be small or large. Change requests that are likely to require a significant effort are typically handled in the same manner as system requests: they follow the same process as the project described in this book, starting with project initiation in Chapter 3 and following through installation in this chapter. Minor changes typically follow a “smaller” version of this same process. There is an initial assessment of feasibility and of costs and benefits, and the change request is prioritized. Then a systems analyst (or a programmer/analyst) performs the analysis, which may include interviewing users, and prepares an initial design before programming begins. The new (or revised) program is then extensively tested before the system is converted from the old system to the revised one.

Change requests typically come from five sources. The most common source is problem reports from the operations group that identify bugs in the system that must be fixed. These are usually given immediate priority because a bug can cause significant problems. Even a minor bug can cause major problems by upsetting users and reducing their acceptance of and confidence in the system.

The second most common source of change requests is enhancement to the system from users. As users work with the system, they often identify minor changes in the design that can make the system easier to use or identify additional functions that are needed. Such enhancements are important in satisfying the users and are often key in ensuring that the system changes as the business requirements change. Enhancements are often given second priority after bug fixes.

CONCEPTS

15-F Software Bugs

IN ACTION

The awful truth is that every operating system and application system is defective. System complexity, the competitive pressure to hurry applications to market, and simple incompetence contribute to the problem.

Will software ever be bug free? Not likely. Microsoft Windows Group General Manager Chris Jones believes that bigger programs breed more bugs. Each revision is usually bigger and more complex than its predecessor, which means new places for bugs to hide. Former Microsoft product manager Richard Freedman agrees that the potential for defects increases as software becomes more complex, but he believes users ultimately win more than they lose. “I’d say the features have gotten exponentially better, and the product quality has degraded a fractional amount.”

Still, the majority of users who responded to our survey said they’d buy a software program with fewer features if it were bug free. This sentiment runs counter to what most software developers believe. “People buy features, plain and simple,” explains Freedman. “There have been attempts to release stripped-down word processors and spreadsheets, and they don’t sell.” Freedman says a trend toward smaller, less-bug-prone software with fewer features will “never happen.”

Eventually, the software ships and the bug reports start rolling in. What happens next is what separates the companies you want to patronize from the slackers. While almost every vendor provides bug fixes eventually, some companies do a better job of it than others. Some observers view Microsoft’s market dominance as a roadblock to bug-free software. Todd Paglia, an attorney with the Washington, D.C.–based Consumer Project on Technology, says, “If actual competition for operating systems existed and we had greater competition for some of the software that runs on the Microsoft operating system, we would have higher quality than we have now.”

Source: “Software Bugs Run Rampant,” *PC World* 17, no. 1 (January 1999): 46.

Question:

1. If commercial systems contain the amount of bugs that the above article suggests, what are the implications for systems developed in-house? Would in-house systems be more likely to have a lower or higher quality than commercial systems? Explain.

A third source of change requests is other system development projects. For example, as part of CD Selections' Internet sales system project, CD Selections likely had to make some minor changes to the distribution system to ensure that the two systems would work together. These changes required by the need to integrate two systems are generally rare but are becoming more common as system integration efforts become more common.

A fourth source of change requests are those that occur when underlying software or networks change. For example, new versions of Windows often require an application to change the way it interacts with Windows, or enables application systems to take advantage of new features that improve efficiency. While users may never see these changes (because most changes are inside the system and do not affect its user interface or functionality), these changes can be among the most challenging to implement because analysts and programmers must learn about the new system characteristics, understand how application systems use (or can use) those characteristics, and then make the needed programming changes.

The fifth source of change requests is senior management. These change requests are often driven by major changes in the organization's strategy (e.g., the CD Selections Internet Sales project) or operations. These significant change requests are typically treated as separate projects, but the project manager responsible for the initial system is often placed in charge of the new project.

Project Assessment

The goal of *project assessment* is to understand what was successful about the system and the project activities (and therefore should be continued in the next system or project) and what needs to be improved. Project assessment is not routine in most organizations, except for military organizations, which are accustomed to preparing after-action reports. Nonetheless, assessment can be an important component in organizational learning because it helps organizations and people understand how to improve their work. It is particularly important for junior staff members because it helps promote faster learning. There are two primary parts to project assessment—project team review and system review.

Project Team Review A *project team review* focuses on the way in which the project team carried out its activities. Each project member prepares a short two- to three-page document that reports and analyzes his or her performance. The focus is on performance improvement, not penalties for mistakes made. By explicitly identifying mistakes and understanding their causes, project team members will, it is hoped, be better prepared for the next time they encounter a similar situation—and less likely to repeat the same mistakes. Likewise, by identifying excellent performance, team members will be able to understand why their actions worked well and how to repeat them in future projects.

The documents prepared by each team member are assessed by the project manager, who meets with the team members to help them understand how to improve their performance. The project manager then prepares a summary document that outlines the key learnings from the project. This summary identifies what actions should be taken in future projects to improve performance but is careful not to identify team members who made mistakes. The summary is widely circulated among all project managers to help them understand how to manage their projects better. Often, it is also circulated among regular staff members who did not work on the project so that they, too, can learn from other projects.

System Review The focus of the *system review* is to understand the extent to which the proposed costs and benefits from the new system that were identified during

project initiation were actually recognized from the implemented system. Project team review is usually conducted immediately after the system is installed, while key events are still fresh in team members' minds, but system review is often undertaken several months after the system is installed because it often takes a while before the system can be properly assessed.

System review starts with the system request and feasibility analysis prepared at the start of the project. The detailed analyses prepared for the expected business value (both tangible and intangible) as well as the economic feasibility analysis are reexamined and a new analysis is prepared after the system has been installed. The objective is to compare the anticipated business value against the actual realized business value from the system. This helps the organization assess whether the system actually provided the value it was planned to provide. Whether or not the system provides the expected value, future projects can benefit from an improved understanding of the true costs and benefits.

A formal system review also has important behavior implications for project initiation. Since everyone involved with the project knows that all statements about business value and the financial estimates prepared during project initiation will be evaluated at the end of the project, they have an incentive to be conservative in their assessments. No one wants to be the project sponsor or project manager for a project that goes radically over budget or fails to deliver promised benefits.

APPLYING THE CONCEPTS AT CD SELECTIONS

Installation of the Internet Sales System at CD Selections was somewhat simpler than the installation of most systems because the system was entirely new; there was no As-Is system for the new system to replace. Also there was not a large number of staff members who needed to be trained on the operation of the new system.

PRACTICAL

TIP

15-1 Beating Buggy Software



How do you avoid bugs in the commercial software you buy? Here are six tips:

1. **Know your software:** Find out if the few programs you use day in and day out have known bugs and patches, and track the Web sites that offer the latest information on them.
2. **Back up your data:** This dictum should be tattooed on every monitor. Stop reading right now and copy the data you can't afford to lose onto a floppy disk, second hard disk, or Web server. We'll wait.
3. **Don't upgrade—yet:** It's tempting to upgrade to the latest and greatest version of your favorite software, but why chance it? Wait a few months, check out other users' experiences with the upgrade on Usenet

newsgroups or the vendor's own discussion forum, and then go for it. But only if you must.

4. **Upgrade slowly:** If you decide to upgrade, allow yourself at least a month to test the upgrade on a separate system before you install it on all the computers in your home or office.
5. **Forget the betas:** Installing beta software on your primary computer is a game of Russian roulette. If you really have to play with beta software, get a second computer.
6. **Complain:** The more you complain about bugs and demand remedies, the more costly it is for vendors to ship buggy products. It's like voting—the more people participate, the better the results.

Source: "Software Bugs Run Rampant," *PC World* 17, no. 1 (January 1999): 46.

Conversion

Conversion went smoothly. First, the new hardware was purchased and installed. Then the software was installed on the Web server and on the client computers to be used by the staff of the Internet sales group. There was no data conversion per se, although the system started receiving data downloaded from the distribution system every day as it would during normal operations.

Alec Adams, senior systems analyst and project manager for CD Selections' Internet sales system, decided on a direct conversion (because there was no As-Is system) in the one location (because there was only one location) of all system modules. The conversion, if you could call it that, went smoothly through the alpha and beta tests described in Chapter 14, and the system was declared technically ready for operation.

Change Management

There were few change management issues because there were no existing staff members who had to change. All new staff were hired, most by internal transfer from other groups within CD Selections. The most likely stakeholders to be concerned by the change would be managers and employees in the traditional retail stores who might see the Internet sales system as a threat to their stores. Alec therefore developed an information campaign (distributed through the employee newsletter and internal Web site) that discussed the reasons for the change and explained that the Internet sales system was seen as a complement to the existing stores, not as a competitor. The system was instead targeted at Web-based competitors, such as Amazon.com and CDnow.

The new management policies were developed, along with a training plan that encompassed both the manual work procedures and computerized procedures. Alec decided to use classroom training for the Internet sales system personnel because there was a small number of them and it was simpler and more cost effective to train them all together in one classroom session.

Post-Implementation Activities

Support of the system was turned over to the CD Selections operations group, who had hired four additional support staff members with expertise in networking and Web-based systems. System maintenance began almost immediately, with Alec designated as the project manager responsible for maintenance of this version of the system plus the development of the next version. Alec began the planning to develop the next version of the system.

Project team review uncovered several key learnings, mostly involving Web-based programming and the difficulties in linking to existing Structured Query Language (SQL) databases. The project was delivered on budget (see Figure 3-13 in Chapter 3), with the exception that more was spent on programming than was anticipated.

A preliminary system review was conducted after two months of operations. Sales were \$40,000 for the first month and \$60,000 for the second, showing a gradual increase (remember that the goal for the first year of operations was \$1,000,000). Operating expenses averaged \$60,000 per month, a bit higher than the projected average, owing to startup costs and the initial marketing campaign. Nonetheless, Margaret Mooney, vice president of marketing and the project sponsor, was quite pleased. She approved the feasibility study for the follow-on project to develop the second version of the Internet sales system, and Alec began the SDLC all over again.

SUMMARY

Conversion

Conversion, the technical process by which the new system replaces the old system, has three major steps: install hardware, install software, and convert data. Conversion style, the way in which users are switched between the old and new systems, can be via either direct conversion (in which users stop using the old system and immediately begin using the new system) or parallel conversion (in which both systems are operated simultaneously to ensure the new system is operating correctly). Conversion location, what parts of the organization are converted when, can be via a pilot conversion in one location; via a phased conversion, in which locations are converted in stages over time; or via simultaneous conversion, in which all locations are converted at the same time. The system can be converted module by module or as a whole at one time. Parallel and pilot conversion are less risky because they have a greater chance of detecting bugs before the bugs have widespread effect, but parallel conversion can be expensive.

Change Management

Change management is the process of helping people to adopt and adapt to the To-Be system and its accompanying work processes. People resist change for very rational reasons, usually because they perceive the costs to themselves of the new system (and the transition to it) to outweigh the benefits. The first step in the change management plan is to change the management policies (such as standard operating procedures), devise measurements and rewards that support the new system, and allocate resources to support it. The second step is to develop a concise list of costs and benefits to the organization and all relevant stakeholders in the change, which will point out who is likely to support and who is likely to resist the change. The third step is to motivate adoption both by providing information and by using political strategies—using power to induce potential adopters to adopt the new system. Finally, training, whether classroom, one-on-one, or computer based, is essential to enable successful adoption. Training should focus on the primary functions the users will perform and look beyond the system itself to help users integrate the system into their routine work processes.

Post-Implementation Activities

System support is performed by the operations group, which provides on-line and help desk support to the users. System support has both a level 1 support staff, which answers the phone and handles most of the questions, and level 2 support staff, which follows up on challenging problems and sometimes generates change requests for bug fixes. System maintenance responds to change requests (from the system support staff, users, other development project teams, and senior management) to fix bugs and improve the business value of the system. The goal of project assessment is to understand what was successful about the system and the project activities (and therefore should be continued in the next system or project) and what needs to be improved. Project team review focuses on the way in which the project team carried out its activities and usually results in documentation of key lessons learned. System review focuses on understanding the extent to which the proposed costs and benefits from the new system that were identified during project initiation were actually recognized from the implemented system.

KEY TERMS

Change agent	Migration plan	Real costs
Change management	Modular conversion	Refreeze
Change request	Modules	Reluctant adopters
Classroom training	On-demand training	Resistant adopters
Computer-based training (CBT)	One-on-one training	Resource allocation
Conversion	Online support	Rewards
Conversion location	Operations group	Risk
Conversion modules	Parallel conversion	Simultaneous conversion
Conversion strategy	Perceived benefits	Sponsor
Conversion style	Perceived costs	Standard operating procedure (SOP)
Cost	Phased conversion	System maintenance
Direct conversion	Pilot conversion	System request
Frequently asked question (FAQ)	Political strategy	System review
Help desk	Post-implementation	System support
Informational strategy	Potential adopter	Time
Institutionalization	Problem report	Training
Level 1 support	Project assessment	Transition process
Level 2 support	Project team review	Unfreeze
Management policies	Ready adopters	Whole-system conversion
Measurements	Real benefits	

QUESTIONS

- What are the three basic steps in managing organizational change?
- What are the major components of a migration plan?
- Compare and contrast direct conversion and parallel conversion.
- Compare and contrast pilot conversion, phased conversion, and simultaneous conversion.
- Compare and contrast modular conversion and whole-system conversion.
- Explain the tradeoffs among selecting between the types of conversion in questions 3, 4, and 5.
- What are the three key roles in any change management initiative?
- Why do people resist change? Explain the basic model for understanding why people accept or resist change.
- What are the three major elements of management policies that must be considered when implementing a new system?
- Compare and contrast an information change management strategy with a political change management strategy. Is one better than the other?
- Explain the three categories of adopters you are likely to encounter in any change management initiative.
- How should you decide what items to include in your training plan?
- Compare and contrast three basic approaches to training.
- What is the role of the operations group in the systems development life cycle (SDLC)?
- Compare and contrast two major ways of providing system support.
- How is a problem report different from a change request?
- What are the major sources of change requests?
- Why is project assessment important?
- How is project team review different from system review?
- What do you think are three common mistakes that novice analysts make in migrating from the As-Is to the To-Be system?
- Some experts argue that change management is more important than any other part of the SDLC. Do you agree or not? Explain.
- In our experience, change management planning often receives less attention than conversion planning. Why do you think this happens?

EXERCISES

- A. Suppose you are installing a new accounting package in your small business. What conversion strategy would you use? Develop a conversion plan (i.e., technical aspects only).
- B. Suppose you are installing a new room reservation system for your university that tracks which courses are assigned to which rooms. Assume that all the rooms in each building are “owned” by one college or department and only one person in that college or department has permission to assign them. What conversion strategy would you use? Develop a conversion plan (i.e., technical aspects only).
- C. Suppose you are installing a new payroll system in a very large multinational corporation. What conversion strategy would you use? Develop a conversion plan (i.e., technical aspects only).
- D. Consider a major change you have experienced in your life (e.g., taking a new job, starting a new school). Prepare a cost–benefit analysis of the change in terms of both the change and the transition to the change.
- E. Suppose you are the project manager for a new library system for your university. The system will improve the way in which students, faculty, and staff can search for books by enabling them to search over the Web, rather than using only the current text-based system available on the computer terminals in the library. Prepare a cost–benefit analysis of the change in terms of both the change and the transition to the change for the major stakeholders.
- F. Prepare a plan to motivate the adoption of the system in Exercise E.
- G. Prepare a training plan that includes both what you would train and how the training would be delivered for the system in Exercise E.
- H. Suppose you are leading the installation of a new decision support system to help admissions officers manage the admissions process at your university. Develop a change management plan (i.e., organizational aspects only).
- I. Suppose you are the project leader for the development of a new Web-based course registration system for your university that replaces an old system in which students had to go to the coliseum at certain times and stand in line to get permission slips for each course they wanted to take. Develop a migration plan (including both technical conversion and change management).
- J. Suppose you are the project leader for the development of a new airline reservation system that will be used by the airline’s in-house reservation agents. The system will replace the current command-driven system designed in the 1970s that uses terminals. The new system uses PCs with a Web-based interface. Develop a migration plan (including both conversion and change management) for your telephone operators.
- K. Develop a migration plan (including both conversion and change management) for the independent travel agencies who use the airline reservation system described in Exercise J.

MINICASES

- Nancy is the IS department head at MOTO Inc., a human resources management firm. The IS staff at MOTO Inc. completed work on a new client management software system about a month ago. Nancy was impressed with the performance of her staff on this project because the firm had not previously undertaken a project of this scale in-house. One of Nancy’s weekly tasks is to evaluate and prioritize the change requests that have come in for the various applications used by the firm.

Right now, Nancy has five change requests for the client system on her desk. One request is from a system

user who would like some formatting changes made to a daily report produced by the system. Another request is from a user who would like the sequence of menu options changed on one of the system menus to more closely reflect the frequency of use for those options. A third request came in from the Billing Department. This department performs billing through the use of a billing software package. A major upgrade of this software is being planned, and the interface between the client system and the bill system will need to be changed to accommodate the new software’s data structures. The fourth request seems to be a system

bug that occurs whenever a client cancels a contract (a rare occurrence, fortunately). The last request came from Susan, the company president. This request confirms the rumor that MOTO Inc. is about to acquire another new business. The new business specializes in the temporary placement of skilled professional and scientific employees, and represents a new business area for MOTO Inc. The client management software system will need to be modified to incorporate the special client arrangements that are associated with the acquired firm.

How do you recommend that Nancy prioritize these change requests for the client/management system?

2. Sky View Aerial Photography offers a wide range of aerial photographic, video, and infrared imaging services. The company has grown from its early days of snapping pictures of client houses to its current status as a full-service aerial image specialist. Sky View now maintains numerous contracts with various governmental agencies for aerial mapping and surveying work. Sky View has its offices at the airport where its fleet of specially-equipped aircraft are hangared. Sky View contracts with several free-lance pilots and photographers for some of its aerial work and also employs several full-time pilots and photographers.

The owners of Sky View Aerial Photography recently contracted with a systems development consulting firm to develop a new information system for

the business. As the number of contracts, aircraft, flights, pilots, and photographers increased, the company experienced difficulty keeping accurate records of its business activity and the utilization of its fleet of aircraft. The new system will require all pilots and photographers to swipe an ID badge through a reader at the beginning and conclusion of each photo flight, along with recording information about the aircraft used and the client served on that flight. These records would be reconciled against the actual aircraft utilization logs maintained and recorded by the hangar personnel.

The office staff was eagerly awaiting the installation of the new system. Their general attitude was that the system would reduce the number of problems and errors that they encountered and would make their work easier. The pilots, photographers, and hangar staff were less enthusiastic, being unaccustomed to having their activities monitored in this way.

- a. Discuss the factors that may inhibit the acceptance of this new system by the pilots, photographers, and hangar staff.
- b. Discuss how an informational strategy could be used to motivate adoption of the new system at Sky View Aerial Photography.
- c. Discuss how a political strategy could be used to motivate adoption of the new system at Sky View Aerial Photography.