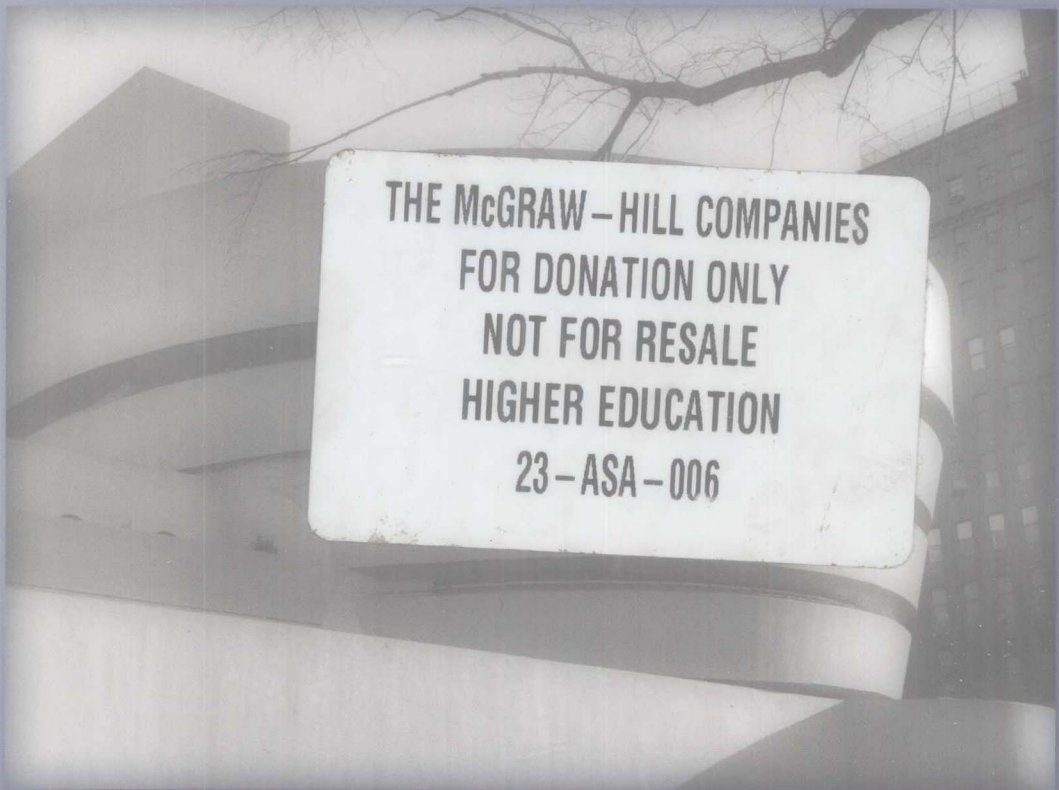


OBJECT-ORIENTED & CLASSICAL SOFTWARE ENGINEERING

SIXTH EDITION



STEPHEN R. SCHACH

Object-Oriented and Classical Software Engineering

Sixth Edition

Stephen R. Schach
Vanderbilt University



Higher Education

Boston Burr Ridge, IL Dubuque, IA Madison, WI New York San Francisco St. Louis
Bangkok Bogotá Caracas Kuala Lumpur Lisbon London Madrid Mexico City
Milan Montreal New Delhi Santiago Seoul Singapore Sydney Taipei Toronto



Higher Education

OBJECT-ORIENTED AND CLASSICAL SOFTWARE ENGINEERING, SIXTH EDITION

Published by McGraw-Hill, a business unit of The McGraw-Hill Companies, Inc., 1221 Avenue of the Americas, New York, NY 10020. Copyright © 2005, 2002, 1999, 1996, 1993, 1990 by The McGraw-Hill Companies, Inc. All rights reserved. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written consent of The McGraw-Hill Companies, Inc., including, but not limited to, in any network or other electronic storage or transmission, or broadcast for distance learning.

Some ancillaries, including electronic and print components, may not be available to customers outside the United States.

This book is printed on acid-free paper.

2 3 4 5 6 7 8 9 0 DOC/DOC 0 9 8 7 6 5 4

ISBN 0-07-286551-2

Publisher: *Elizabeth A. Jones*
Managing developmental editor: *Emily J. Lupash*
Marketing manager: *Dawn R. Bercier*
Senior project manager: *Jane Mohr*
Lead production supervisor: *Sandy Ludovissy*
Lead media project manager: *Audrey A. Reiter*
Senior coordinator of freelance design: *Michelle D. Whitaker*
Cover designer: *Christopher Reese*
Cover image: © *Sherman/Getty Images*
Compositor: *Interactive Composition Corporation*
Typeface: *10/12 Times Roman*
Printer: *R. R. Donnelley Crawfordsville, IN*

Library of Congress Cataloging-in-Publication Data

Schach, Stephen R.

Object-oriented and classical software engineering / Stephen R. Schach. — 6th ed.

p. cm.

Includes bibliographical references and index.

ISBN 0-07-286551-2

1. Software engineering. 2. Object-oriented programming (Computer science). 3. UML (Computer science).
4. C++ (Computer program language). I. Title.

QA76.758S318 2005

005.1'17—dc22

2003024034

CIP

The following are registered trademarks:

ADF	JBuilder	Requisite Pro
Analyst/Designer	Linux	Rhapsody
Ant	Lotus 1-2-3	Rose
Apache	Lucent Technologies	SilkTest
Apple	MacApp	Software through Pictures
AS/400	Macintosh	Solaris
AT&T	Macintosh Toolbox	SourceSafe
Bachman Product Set	MacProject	SPARCstation
Borland	Microsoft Foundation	Sun
Bugzilla	Class Library	Sun Enterprise
Capability Maturity Model	Motif	Sun Microsystems
CCC	MS-DOS	Sun ONE Studio
ClearQuest	MVS/360	System Architect
CMM	Natural	Together
Coca-Cola	Netscape	UNIX
CORBA	<i>New York Times</i>	VAX
CVS	Object C	Visual Component Library
DB2	Objective-C	Visual C++
e-Components	ObjectWindows Library	Visual J++
Emeraude	1-800-flowers.com	VM/370
Enterprise JavaBeans	Oracle	VMS
Excel	OS/360	<i>Wall Street Journal</i>
Focus	OS/370	WebSphere
Ford	OS/VS2	Win32
Foundation	Post-it note	Windows 95
FoxBASE	PowerBuilder	Windows 2000
GCC	Project	Windows NT
Hewlett-Packard	PureCoverage	X11
IBM	PVCS	XRunner
IMS/360	QAPartner	Zip disk
Java	Rational	ZIP code

Preface

Since the publication of the fifth edition three years ago, the Unified Process has become widely regarded as the methodology of choice for object-oriented software development. The Unified Process is based on Booch's method, Objectory, and OMT. These three older object-oriented methodologies are now no longer supported by their respective authors. Accordingly, the major new feature of the sixth edition is the inclusion of the Unified Process. In particular, the two running case studies are developed using the Unified Process, so the student is exposed to both the theory and the practice of the Unified Process.

Other Key Features of the Sixth Edition

- Chapter 1 has been totally revised. In particular, the strengths of the object-oriented paradigm are analyzed in greater depth. Also, I have introduced the new definition of maintenance adopted by ISO, IEC, IEEE, and EIA.
- The order of Chapters 2 and 3 has been reversed to introduce the evolution-tree life-cycle model and the iterative-and-incremental life-cycle model as early as possible. However, as with all previous editions, numerous other life-cycle models are presented, compared, and contrasted.
- In Chapter 3, "The Software Process," the workflows (activities) and phases of the Unified Process are introduced, and the need for two-dimensional life-cycle models is explained.
- Chapters 4 through 9 have been updated. For example, component-based software engineering is introduced in Chapter 8, and the new IEEE standard for software project management plans is presented in Chapter 9. An example of a plan that conforms to the new standard appears in Appendix F.
- The material on interoperability has been removed from Chapter 8. In both the fourth and the fifth editions, these sections became hopelessly out of date during the 6 months it took to publish the books. In my opinion, the field is moving too fast to be included in a textbook; an instructor wishing to include interoperability in a software engineering course should obtain up-to-the-minute material from the Internet.
- Chapter 10 ("Requirements"), Chapter 12 ("Object-Oriented Analysis"), and Chapter 13 ("Design") have undergone major changes to incorporate the workflows (activities) of the Unified Process. For obvious reasons, Chapter 11 ("Classical Analysis") has not been changed.
- The material on implementation and integration (Chapters 14 and 15 of the fifth edition) has been merged into the new Chapter 14, but there is still a clear separation between implementation and integration.
- Chapter 15 is now on post-delivery maintenance.
- Chapter 16 is new. The fourth edition was the first software engineering textbook to utilize the Unified Modeling Language (UML), which was introduced shortly before that edition was published. Three years later, UML had been formally standardized and

become so widely used that, in the fifth edition, I continued to employ UML to represent object-oriented analysis and object-oriented design, as well as in diagrams that depicted objects and their interrelationships. In both the fourth and fifth editions, I included sufficient material on UML to enable students to do all the exercises, as well as the team-based term project. However, UML is now such an integral part of software engineering in general (and the Unified Process in particular) that I have added a final chapter, “More on UML.” The purpose of Chapter 16 is to provide additional material on UML to prepare the student even more thoroughly for employment in the software industry. This chapter is of particular use to instructors who utilize this book for the two-semester software engineering course sequence. In the second semester, in addition to developing the team-based term project or a capstone project, the student can acquire additional knowledge of UML, beyond what is needed for this book.

- In addition to the two running case studies that are used to illustrate the complete life cycle, seven mini case studies highlight specific topics, such as the moving target problem, stepwise refinement, and post-delivery maintenance.
- The material on extreme programming (XP) has been expanded. In addition, XP is now described within the context of agile programming. XP still is controversial, but I feel that students need to understand the topic so they can decide for themselves whether XP is merely a current fad or a genuine major breakthrough in software engineering.

Features Retained from the Fifth Edition

The key features of the fifth edition have all been retained.

- In all the previous editions, I stressed the importance of documentation, maintenance, reuse, portability, testing, and CASE tools. In this edition, all these concepts are stressed equally firmly. It is no use teaching students the latest ideas unless they appreciate the importance of the basics of software engineering.
- As in the fifth edition, particular attention is paid to object-oriented life-cycle models, object-oriented analysis, object-oriented design, management implications of the object-oriented paradigm, and the testing and maintenance of object-oriented software. Metrics for the object-oriented paradigm are also included. In addition, many briefer references are made to objects, a paragraph or even only a sentence in length. The reason is that the object-oriented paradigm is not concerned just with how the various phases are performed but with how it permeates the way we think about software engineering. Object technology again pervades this book.
- The software process is still the concept that underlies the book as a whole. To control the process, we have to be able to measure what is happening to the project. Accordingly, the emphasis on metrics is retained. With regard to process improvement, the material on the capability maturity model (CMM), ISO/IEC 15504 (SPICE), and ISO/IEC 12207 has been retained; the people capability maturity model (P-CMM) has been added to the chapter on teams.
- The book is still language independent; the few code examples are presented in C++ and Java, and I have made every effort to smooth over language-dependent details and ensure that the code examples are equally clear to C++ and Java users. For example, instead of using `cout` for C++ output and `System.out.println` for Java output, I have

utilized the pseudocode instruction *print*. (The one exception is the new case study, where complete implementation details are given in both C++ and Java.)

- As before, there are two running case studies. The Osbert Oglesby case study from the fourth edition and the Elevator Problem case study (from previous editions) have been redeveloped using the Unified Process. As usual, Java and C++ implementations are available online at www.mhhe.com/engcs/compsci/schach.
- As in the fifth edition, this book contains over 600 references. I selected current research papers as well as classic articles and books whose message remains fresh and relevant. There is no question that software engineering is a rapidly moving field and students therefore need to know the latest results and where in the literature to find them. At the same time, today's cutting-edge research is based on yesterday's truths, and I see no reason to exclude an older reference if its ideas are as applicable today as they originally were.
- With regard to prerequisites, it is assumed that the reader is familiar with one high-level programming language such as C, C++, Ada, or Java. In addition, the reader is expected to have taken a course in data structures.

Why the Classical Paradigm Still Is Included

When I started writing the sixth edition, I decided to exclude all mention of the classical (structured) paradigm. After all, there is now almost unanimous agreement that the object-oriented paradigm is superior to the classical paradigm. I soon discovered, however, that attempting to eliminate any mention of the classical paradigm was distinctly unwise.

First, it is impossible to appreciate why object-oriented technology is superior to classical technology without fully understanding the classical approach and how it differs from the object-oriented approach. For example, the object-oriented paradigm uses an iterative and incremental life-cycle model. To show why such a life-cycle model is needed, it is essential to explain in detail the differences between classical life-cycle models like the waterfall model and the iterative and incremental life-cycle model of the object-oriented paradigm. Therefore, throughout the book, I have included material on the classical paradigm so that the student can clearly appreciate the differences between the classical paradigm and the object-oriented paradigm.

The second reason why I have included both paradigms is that technology transfer is a slow process. Notwithstanding the impact of Y2K on accelerating the switch to the object-oriented paradigm, the majority of software organizations still have not yet adopted the object-oriented paradigm. It therefore is likely that many of the students who use this book will be employed by organizations that use classical software engineering techniques. Furthermore, even when an organization uses the object-oriented approach for developing new software, existing software still has to be maintained, and this legacy software is not object oriented. Therefore, excluding classical material would be unfair to many of the students who use this text.

A third reason for including both paradigms is that a student who is employed at an organization considering making the transition to object-oriented technology will be able to advise that organization regarding both the strengths and the weaknesses of the new paradigm. So, as in the previous edition, the classical and object-oriented approaches are compared, contrasted, and analyzed.

How the Sixth Edition Is Organized

Like the fifth edition of this book, the sixth edition is written for both the traditional one-semester and the newer two-semester software engineering curriculum. In the traditional one-semester (or one-quarter) course, the instructor has to rush through the theoretical material to provide the students the knowledge and skills needed for the term project as soon as possible. The need for haste is so that the students can commence the term project early enough to complete it by the end of the semester. To cater to a one-semester, project-based software engineering course, Part 2 of this book covers the software life cycle, workflow by workflow, and Part 1 contains the theoretical material needed to understand Part 2. For example, Part 1 introduces the reader to CASE, metrics, and testing; each chapter of Part 2 contains a section on CASE tools for that workflow, a section on metrics for that workflow, and a section on testing during that workflow. Part 1 is kept short to enable the instructor to start Part 2 relatively early in the semester. Furthermore, the last two chapters of Part 1 (Chapters 8 and 9) may be postponed, then taught in parallel with Part 2. As a result, the class can begin developing the term project as soon as possible.

We turn now to the two-semester software engineering curriculum. More and more computer science and computer engineering departments are realizing that the overwhelming preponderance of their graduates find employment as software engineers. As a result, many colleges and universities have introduced a two-semester (or two-quarter) software engineering sequence. The first course is largely theoretical (but almost always there is a small project of some sort). The second course comprises a major team-based term project, usually a capstone project. When the term project is in the second course, there is no need for the instructor to rush to start Part 2.

Therefore, an instructor teaching a one-semester (or one-quarter) sequence using the sixth edition covers most of Chapters 1 through 7, then starts Part 2 (Chapters 10 through 16). Chapters 8 and 9 can then be taught in parallel with Part 2 or at the end of the course while the students are implementing the term project. When teaching the two-semester sequence, the chapters of the book are taught in order; the class now is fully prepared for the team-based term project that they will develop in the following semester.

To ensure that the key software engineering techniques of Part 2 truly are understood, each is presented twice. First, whenever a technique is introduced, it is illustrated by means of the elevator problem. The elevator problem is the correct size for the reader to be able to see the technique applied to a complete problem, and it has enough subtleties to highlight both the strengths and weaknesses of the technique being taught. Then, the relevant portion of the Osbert Oglesby case study is presented. This detailed solution provides the second illustration of each technique.

The Problem Sets

The previous edition had four types of problems. There are now five types of problems. The new type of problem is the running object-oriented analysis and design projects at the end of Chapters 10, 12, and 13. These have been included because the only way to learn how to perform the requirements, analysis, and design workflows is from extensive hands-on experience.

Second, the end of each chapter contains a number of exercises intended to highlight key points. These exercises are self-contained; the technical information for all the exercises can be found in this book.

Third, there is a software term project. It is designed to be solved by students working in teams of three, the smallest number of team members that cannot confer over a standard telephone. The term project comprises 16 separate components, each tied to the relevant chapter. For example, design is the topic of Chapter 13, so in that chapter the component of the term project is concerned with software design. By breaking a large project into smaller, well-defined pieces, the instructor can monitor the progress of the class more closely. The structure of the term project is such that an instructor may freely apply the 16 components to any other project that he or she chooses.

Because this book is written for use by graduate students as well as upper-class undergraduates, the fourth type of problem is based on research papers in the software engineering literature. In each chapter, an important paper has been chosen; wherever possible, a paper related to object-oriented software engineering has been selected. The student is asked to read the paper and answer a question relating its contents. Of course, the instructor is free to assign any other research paper; the For Further Reading section at the end of each chapter includes a wide variety of relevant papers.

The fifth type of problem relates to the Osbert Oglesby case study. This type of problem was first introduced in the third edition in response to a number of instructors who feel that their students learn more by modifying an existing product than by developing a new product from scratch. Many senior software engineers in the industry agree with that viewpoint. Accordingly, each chapter in which the case study is presented has at least three problems that require the student to modify the case study in some way. For example, in one chapter the student is asked to redesign the case study using a different design technique from the one used for the case study. In another chapter, the student is asked what the effect would have been of performing the steps of the object-oriented analysis in a different order. To make it easy to modify the source code of the case study, it is available on the World Wide Web at www.mhhe.com/engcs/compSci/schach. The website also has a complete set of PowerPoint lecture notes.

The *Instructor's Solution Manual* contains detailed solutions to all the exercises, as well as to the term project. The *Instructor's Solution Manual* is available from McGraw-Hill.

Acknowledgments

I greatly appreciate the constructive criticisms and many helpful suggestions of the reviewers of the five previous editions, including

Arvin Agah

University of Kansas

Kiumi Akingbehin

University of Michigan, Dearborn

Phil Bernhard

Clemson University

Dan Berry

The Technion

Don Bickerstaff

Eastern Washington University

Richard J. Botting

California State University, San Bernardino

James Cardow

Air Force Institute of Technology

Betty Cheng

Michigan State University

David Cheriton

Stanford University

Thaddeus R. Crews, Jr.

Western Kentucky University

Buster Dunsmore

Purdue University

Eduardo B. Fernandez

Florida Atlantic University

Michael Godfrey

Cornell University

Bob Goldberg

IBM

Donald Gotterbarn

East Tennessee State University

Scott Hawker

University of Alabama

Thomas B. Horton

Florida Atlantic University

Greg Jones

Utah State University

Peter E. Jones

University of Western Australia

Gail Kaiser

Columbia University

Laxmikant V. Kale

University of Illinois

Helene Kershner

University of Buffalo

Chung Lee

California State Polytechnic, Pomona

Richar A. Lejk

University of North Carolina, Chapel Hill

Bill McCracken

Georgia Institute of Technology

Susan Mengel

Texas Tech University

Everald E. Mills

Seattle University

Fred Mowle

Purdue University

Ron New

Johns Hopkins University

David Notkin

University of Washington

Hal Render

University of Colorado, Colorado Springs

David S. Rosenblum

University of California, Irvine

Shmuel Rotenstreich

George Washington University

Wendel Scarborough

Azusa Pacific University

Bob Schuerman

State College, Pennsylvania

Gerald B. Sheble

Iowa State

K. C. Tai

North Carolina State University

Toby Teorey

University of Michigan

Jie We

City University of New York

Laurie Werth

University of Texas, Austin

Lee White

Case Western Reserve University

David Workman

University of Central Florida

George W. Zobrist

University of Missouri, Rolla

In addition, special thanks go to the reviewers of this edition, including

Michael Buckley

State University New York, Buffalo

Catherine Lowry Campbell

New Jersey Institute of Technology

Frances Grodzinsky

Sacred Heart University

Jim Han

Florida Atlantic University

Werner Krandick*Drexel University***Owen Lavin***DePaul University***Donald Needham***United States Naval Academy***Andy Podgurski***Case Western Reserve University***David C. Rine***George Mason University***Mansur Samadzadeh***Oklahoma State University***John H. Saylor***University of Michigan***Fred Strauss***Polytechnic University*

All the above reviewers, without exception, have made significant contributions. Nevertheless, I am particularly grateful to Owen Lavin of DePaul University for his many helpful suggestions for improving this book.

I warmly thank three individuals who have also made contributions to earlier books. First, Jeff Gray implemented the Osbert Oglesby case study. Second, Kris Irwin provided a complete solution to the term project, including implementing it in both Java and C++. Third, my daughter Lauren was again a coauthor of the *Instructor's Solution Manual* and contributor to the PowerPoint slides.

Turning now to my publisher, McGraw-Hill, as always I am most grateful to my publisher Betsy Jones and my developmental editor Emily Lupash for their assistance and guidance from start to finish. I thank copyeditor Gnomi Schrift Gouldin for once again providing so many effective suggestions. It was a pleasure to work with marketing manager Dawn Bercier, senior freelance design coordinator Michelle Whitaker, and lead media project manager Audrey Reiter. Finally, I particularly wish to thank senior project manager Jane Mohr for her boundless help and support. No problem was too big or too small for her to solve.

As always, I enjoyed working with the compositors at Interactive Composition Corporation. Amy Rose was a most able project manager.

I would like to thank the many instructors from all over the world who sent me e-mail regarding the fifth edition. I am exceedingly appreciative of their suggestions, comments, and criticisms. I look forward with anticipation to receiving instructors' feedback on this edition also. My e-mail address is srs@vuse.vanderbilt.edu.

Students, too, have been most helpful. I thank my students at Vanderbilt University for their numerous questions and comments, both inside and outside the classroom. In particular, I would like to express my appreciation to Paul Bielaczyc, Zhihong Ding, Shaivya Easwaren, Kenon Ewing, Sarita Gupta, Julia Irani, Andrews Jebasingh, Pavil Jose, Anantha Narayanan, Joshua Phillips, Adam Loeb Small, Larry Thomas, HariPriya Venkatesan, and Bin Zhou, the students in a graduate course on the Unified Process that I recently taught at Vanderbilt. I am truly grateful for their insights and creative ideas.

I am also most appreciative of the provocative questions and constructive suggestions e-mailed to me by students from all over the world. As with the previous editions, I look forward keenly to student feedback on this edition, too.

Finally, as always, I thank my family for their continual support. When I started writing books, my limited free time had to be shared between my young children and my current book project. Now that my children are adults and work with me on my books, writing has become a family activity. For the twelfth time, it is my privilege to dedicate this book to my wife, Sharon, and my children, David and Lauren, with love.

Stephen R. Schach

Contents

Preface xv

PART ONE

INTRODUCTION TO SOFTWARE ENGINEERING 1

Chapter 1

The Scope of Software Engineering 3

- Learning Objectives 3
- 1.1 Historical Aspects 4
- 1.2 Economic Aspects 6
- 1.3 Maintenance Aspects 7
 - 1.3.1 *Classical and Modern Views of Maintenance* 9
 - 1.3.2 *The Importance of Postdelivery Maintenance* 11
- 1.4 Requirements, Analysis, and Design Aspects 13
- 1.5 Team Development Aspects 15
- 1.6 Why There Is No Planning Phase 16
- 1.7 Why There Is No Testing Phase 17
- 1.8 Why There Is No Documentation Phase 17
- 1.9 The Object-Oriented Paradigm 18
- 1.10 The Object-Oriented Paradigm in Perspective 23
- 1.11 Terminology 23
- 1.12 Ethical Issues 26
- Chapter Review 27
- For Further Reading 27
- Key Terms 28
- Problems 29
- References 30

Chapter 2

Software Life-Cycle Models 34

- Learning Objectives 34
- 2.1 Software Development in Theory 34
- 2.2 Winburg Mini Case Study 35

- 2.3 Lessons of the Winburg Mini Case Study 39
- 2.4 Teal Tractors Mini Case Study 39
- 2.5 Iteration and Incrementation 40
- 2.6 Winburg Mini Case Study Revisited 44
- 2.7 Risks and Other Aspects of Iteration and Incrementation 45
- 2.8 Managing Iteration and Incrementation 47
- 2.9 Other Life-Cycle Models 48
 - 2.9.1 *Code-and-Fix Life-Cycle Model* 48
 - 2.9.2 *Waterfall Life-Cycle Model* 49
 - 2.9.3 *Rapid-Prototyping Life-Cycle Model* 51
 - 2.9.4 *Extreme Programming and Agile Processes* 52
 - 2.9.5 *Synchronize-and-Stabilize Life-Cycle Model* 54
 - 2.9.6 *Spiral Life-Cycle Model* 54
- 2.10 Comparison of Life-Cycle Models 58
- Chapter Review 59
- For Further Reading 60
- Key Terms 60
- Problems 61
- References 61

Chapter 3

The Software Process 64

- Learning Objectives 64
- 3.1 The Unified Process 66
- 3.2 Iteration and Incrementation within the Object-Oriented Paradigm 67
- 3.3 The Requirements Workflow 68
- 3.4 The Analysis Workflow 70
- 3.5 The Design Workflow 72
- 3.6 The Implementation Workflow 73
- 3.7 The Test Workflow 74
 - 3.7.1 *Requirements Artifacts* 74
 - 3.7.2 *Analysis Artifacts* 74
 - 3.7.3 *Design Artifacts* 75
 - 3.7.4 *Implementation Artifacts* 75

3.8	Postdelivery Maintenance	77
3.9	Retirement	78
3.10	The Phases of the Unified Process	78
3.10.1	<i>The Inception Phase</i>	79
3.10.2	<i>The Elaboration Phase</i>	81
3.10.3	<i>The Construction Phase</i>	82
3.10.4	<i>The Transition Phase</i>	82
3.11	One- versus Two-Dimensional Life-Cycle Models	83
3.12	Improving the Software Process	84
3.13	Capability Maturity Models	85
3.14	Other Software Process Improvement Initiatives	88
3.15	Costs and Benefits of Software Process Improvement	89
	Chapter Review	91
	For Further Reading	91
	Key Terms	92
	Problems	92
	References	93

Chapter 4

Teams 96

	Learning Objectives	96
4.1	Team Organization	96
4.2	Democratic Team Approach	98
4.2.1	<i>Analysis of the Democratic Team Approach</i>	99
4.3	Classical Chief Programmer Team Approach	99
4.3.1	<i>The New York Times Project</i>	101
4.3.2	<i>Impracticality of the Classical Chief Programmer Team Approach</i>	102
4.4	Beyond Chief Programmer and Democratic Teams	102
4.5	Synchronize-and-Stabilize Teams	106
4.6	Extreme Programming Teams	106
4.7	People Capability Maturity Model	107
4.8	Choosing an Appropriate Team Organization	108
	Chapter Review	109
	For Further Reading	109
	Key Terms	109
	Problems	109
	References	110

Chapter 5

The Tools of the Trade 112

	Learning Objectives	112
5.1	Stepwise Refinement	112
5.1.1	<i>Stepwise Refinement Mini Case Study</i>	113
5.2	Cost–Benefit Analysis	118
5.3	Software Metrics	119
5.4	CASE	121
5.5	Taxonomy of CASE	122
5.6	Scope of CASE	123
5.7	Software Versions	127
5.7.1	<i>Revisions</i>	127
5.7.2	<i>Variations</i>	128
5.8	Configuration Control	129
5.8.1	<i>Configuration Control during Postdelivery Maintenance</i>	131
5.8.2	<i>Baselines</i>	131
5.8.3	<i>Configuration Control during Development</i>	132
5.9	Build Tools	132
5.10	Productivity Gains with CASE Technology	133
	Chapter Review	135
	For Further Reading	135
	Key Terms	135
	Problems	136
	References	137

Chapter 6

Testing 139

	Learning Objectives	139
6.1	Quality Issues	140
6.1.1	<i>Software Quality Assurance</i>	141
6.1.2	<i>Managerial Independence</i>	141
6.2	Non-Execution-Based Testing	142
6.2.1	<i>Walkthroughs</i>	143
6.2.2	<i>Managing Walkthroughs</i>	143
6.2.3	<i>Inspections</i>	144
6.2.4	<i>Comparison of Inspections and Walkthroughs</i>	146
6.2.5	<i>Strengths and Weaknesses of Reviews</i>	147
6.2.6	<i>Metrics for Inspections</i>	147

6.3	Execution-Based Testing	147
6.4	What Should Be Tested?	148
	6.4.1 Utility	149
	6.4.2 Reliability	149
	6.4.3 Robustness	149
	6.4.4 Performance	150
	6.4.5 Correctness	150
6.5	Testing versus Correctness Proofs	152
	6.5.1 Example of a Correctness Proof	152
	6.5.2 Correctness Proof Mini Case Study	156
	6.5.3 Correctness Proofs and Software Engineering	157
6.6	Who Should Perform Execution-Based Testing?	159
6.7	When Testing Stops	161
	Chapter Review	161
	For Further Reading	161
	Key Terms	162
	Problems	162
	References	164

Chapter 7

From Modules to Objects 166

	Learning Objectives	166
7.1	What Is a Module?	166
7.2	Cohesion	170
	7.2.1 Coincidental Cohesion	170
	7.2.2 Logical Cohesion	171
	7.2.3 Temporal Cohesion	172
	7.2.4 Procedural Cohesion	172
	7.2.5 Communicational Cohesion	173
	7.2.6 Functional Cohesion	173
	7.2.7 Informational Cohesion	174
	7.2.8 Cohesion Example	174
7.3	Coupling	175
	7.3.1 Content Coupling	176
	7.3.2 Common Coupling	176
	7.3.3 Control Coupling	178
	7.3.4 Stamp Coupling	178
	7.3.5 Data Coupling	180
	7.3.6 Coupling Example	180
	7.3.7 The Importance of Coupling	181

7.4	Data Encapsulation	182
	7.4.1 Data Encapsulation and Development	184
	7.4.2 Data Encapsulation and Maintenance	185
7.5	Abstract Data Types	191
7.6	Information Hiding	192
7.7	Objects	194
7.8	Inheritance, Polymorphism, and Dynamic Binding	198
7.9	The Object-Oriented Paradigm	200
	Chapter Review	203
	For Further Reading	203
	Key Terms	204
	Problems	204
	References	205

Chapter 8

Reusability and Portability 208

	Learning Objectives	208
8.1	Reuse Concepts	209
8.2	Impediments to Reuse	211
8.3	Reuse Case Studies	212
	8.3.1 Raytheon Missile Systems Division	212
	8.3.2 European Space Agency	214
8.4	Objects and Reuse	215
8.5	Reuse during Design and Implementation	215
	8.5.1 Design Reuse	215
	8.5.2 Application Frameworks	217
	8.5.3 Design Patterns	217
	8.5.4 Software Architecture	220
	8.5.5 Component-Based Software Engineering	222
8.6	Reuse and Postdelivery Maintenance	222
8.7	Portability	223
	8.7.1 Hardware Incompatibilities	224
	8.7.2 Operating System Incompatibilities	225
	8.7.3 Numerical Software Incompatibilities	225
	8.7.4 Compiler Incompatibilities	226
8.8	Why Portability?	229

- 8.9** Techniques for Achieving Portability 230
 - 8.9.1 *Portable System Software* 230
 - 8.9.2 *Portable Application Software* 231
 - 8.9.3 *Portable Data* 232
- Chapter Review 233
- For Further Reading 233
- Key Terms 234
- Problems 234
- References 236

Chapter 9 Planning and Estimating 240

- Learning Objectives 240
- 9.1** Planning and the Software Process 241
- 9.2** Estimating Duration and Cost 242
 - 9.2.1 *Metrics for the Size of a Product* 243
 - 9.2.2 *Techniques of Cost Estimation* 247
 - 9.2.3 *Intermediate COCOMO* 249
 - 9.2.4 *COCOMO II* 252
 - 9.2.5 *Tracking Duration and Cost Estimates* 253
- 9.3** Components of a Software Project Management Plan 254
- 9.4** Software Project Management Plan Framework 255
- 9.5** IEEE Software Project Management Plan 257
- 9.6** Planning Testing 260
- 9.7** Planning Object-Oriented Projects 261
- 9.8** Training Requirements 261
- 9.9** Documentation Standards 262
- 9.10** CASE Tools for Planning and Estimating 263
- 9.11** Testing the Software Project Management Plan 263
 - Chapter Review 263
 - For Further Reading 264
 - Key Terms 264
 - Problems 265
 - References 266

PART TWO THE WORKFLOWS OF THE SOFTWARE LIFE CYCLE 269

Chapter 10 Requirements 271

- Learning Objectives 271
- 10.1** Determining What the Client Needs 271
- 10.2** Overview of the Requirements Workflow 272
- 10.3** Understanding the Domain 273
- 10.4** The Business Model 274
 - 10.4.1 *Interviewing* 274
 - 10.4.2 *Other Techniques* 275
 - 10.4.3 *Use Cases* 276
- 10.5** Initial Requirements 277
- 10.6** Initial Understanding of the Domain: The Osbert Oglesby Case Study 278
- 10.7** Initial Business Model: The Osbert Oglesby Case Study 279
- 10.8** Initial Requirements: The Osbert Oglesby Case Study 282
- 10.9** Continuing the Requirements Workflow: The Osbert Oglesby Case Study 284
- 10.10** The Test Workflow: The Osbert Oglesby Case Study 291
- 10.11** The Classical Requirements Phase 292
- 10.12** Rapid Prototyping 293
- 10.13** Human Factors 294
- 10.14** Reusing the Rapid Prototype 295
- 10.15** CASE Tools for the Requirements Workflow 296
- 10.16** Metrics for the Requirements Workflow 297
- 10.17** Challenges of the Requirements Workflow 297
 - Chapter Review 299
 - For Further Reading 299
 - Key Terms 300
 - Case Study Key Terms 300
 - Problems 300
 - References 301

Chapter 11

Classical Analysis 303

- Learning Objectives 303
- 11.1 The Specification Document 303
- 11.2 Informal Specifications 305
 - 11.2.1 *Correctness Proof Mini Case Study Redux* 306
- 11.3 Structured Systems Analysis 307
 - 11.3.1 *Sally's Software Shop Mini Case Study* 307
- 11.4 Structured Systems Analysis: The Osbert Oglesby Case Study 315
- 11.5 Other Semiformal Techniques 316
- 11.6 Entity-Relationship Modeling 317
- 11.7 Finite State Machines 319
 - 11.7.1 *Finite State Machines: The Elevator Problem Case Study* 321
- 11.8 Petri Nets 325
 - 11.8.1 *Petri Nets: The Elevator Problem Case Study* 328
- 11.9 Z 330
 - 11.9.1 *Z: The Elevator Problem Case Study* 330
 - 11.9.2 *Analysis of Z* 333
- 11.10 Other Formal Techniques 334
- 11.11 Comparison of Classical Analysis Techniques 335
- 11.12 Testing during Classical Analysis 335
- 11.13 CASE Tools for Classical Analysis 336
- 11.14 Metrics for Classical Analysis 337
- 11.15 Software Project Management Plan: The Osbert Oglesby Case Study 338
- 11.16 Challenges of Classical Analysis 338
 - Chapter Review 339
 - For Further Reading 339
 - Key Terms 340
 - Case Study Key Terms 340
 - Problems 340
 - References 342

Chapter 12

Object-Oriented Analysis 346

- Learning Objectives 346
- 12.1 The Analysis Workflow 347

- 12.2 Extracting the Entity Classes 348
- 12.3 Object-Oriented Analysis: The Elevator Problem Case Study 349
- 12.4 Functional Modeling: The Elevator Problem Case Study 349
- 12.5 Entity Class Modeling: The Elevator Problem Case Study 351
 - 12.5.1 *Noun Extraction* 352
 - 12.5.2 *CRC Cards* 354
- 12.6 Dynamic Modeling: The Elevator Problem Case Study 355
- 12.7 The Test Workflow: Object-Oriented Analysis 358
- 12.8 Extracting the Boundary and Control Classes 362
- 12.9 The Initial Functional Model: The Osbert Oglesby Case Study 363
- 12.10 The Initial Class Diagram: The Osbert Oglesby Case Study 365
- 12.11 The Initial Dynamic Model: The Osbert Oglesby Case Study 371
- 12.12 Extracting the Boundary Classes: The Osbert Oglesby Case Study 373
- 12.13 Extracting the Control Classes: The Osbert Oglesby Case Study 374
- 12.14 Refining the Use Cases: The Osbert Oglesby Case Study 374
- 12.15 Use-Case Realization: The Osbert Oglesby Case Study 377
 - 12.15.1 *Buy a Masterpiece Use Case* 377
 - 12.15.2 *Buy a Masterwork Use Case* 382
 - 12.15.3 *Buy Other Painting Use Case* 383
 - 12.15.4 *The Remaining Five Use Cases* 384
- 12.16 Incrementing the Class Diagram: The Osbert Oglesby Case Study 386
- 12.17 The Test Workflow: The Osbert Oglesby Case Study 388
- 12.18 The Specification Document in the Unified Process 389
- 12.19 More on Actors and Use Cases 390
- 12.20 CASE Tools for the Object-Oriented Analysis Workflow 391

- 12.21** Challenges of the Object-Oriented Analysis Workflow 391
- Chapter Review 392
- For Further Reading 392
- Key Terms 393
- Problems 393
- References 395

Chapter 13 Design 397

- Learning Objectives 397
- 13.1** Design and Abstraction 398
- 13.2** Operation-Oriented Design 398
- 13.3** Data Flow Analysis 399
 - 13.3.1 Mini Case Study: Word Counting 400
 - 13.3.2 Data Flow Analysis Extensions 405
- 13.4** Transaction Analysis 405
- 13.5** Data-Oriented Design 407
- 13.6** Object-Oriented Design 408
- 13.7** Object-Oriented Design: The Elevator Problem Case Study 409
- 13.8** Object-Oriented Design: The Osbert Oglesby Case Study 412
- 13.9** The Design Workflow 416
- 13.10** The Test Workflow: Design 418
- 13.11** The Test Workflow: The Osbert Oglesby Case Study 419
- 13.12** Formal Techniques for Detailed Design 419
- 13.13** Real-Time Design Techniques 419
- 13.14** CASE Tools for Design 421
- 13.15** Metrics for Design 421
- 13.16** Challenges of the Design Workflow 422
 - Chapter Review 423
 - For Further Reading 423
 - Key Terms 424
 - Problems 424
 - References 425

Chapter 14 Implementation 428

- Learning Objectives 428
- 14.1** Choice of Programming Language 428
- 14.2** Fourth-Generation Languages 431

- 14.3** Good Programming Practice 433
 - 14.3.1 Use of Consistent and Meaningful Variable Names 434
 - 14.3.2 The Issue of Self-Documenting Code 435
 - 14.3.3 Use of Parameters 436
 - 14.3.4 Code Layout for Increased Readability 437
 - 14.3.5 Nested **if** Statements 437
- 14.4** Coding Standards 439
- 14.5** Code Reuse 439
- 14.6** Integration 440
 - 14.6.1 Top-down Integration 440
 - 14.6.2 Bottom-up Integration 442
 - 14.6.3 Sandwich Integration 442
 - 14.6.4 Integration of Object-Oriented Products 443
 - 14.6.5 Management of Integration 444
- 14.7** The Implementation Workflow 445
- 14.8** The Implementation Workflow: The Osbert Oglesby Case Study 445
- 14.9** The Test Workflow: Implementation 446
- 14.10** Test Case Selection 446
 - 14.10.1 Testing to Specifications versus Testing to Code 446
 - 14.10.2 Feasibility of Testing to Specifications 446
 - 14.10.3 Feasibility of Testing to Code 447
- 14.11** Black-Box Unit-Testing Techniques 450
 - 14.11.1 Equivalence Testing and Boundary Value Analysis 450
 - 14.11.2 Functional Testing 451
- 14.12** Black-Box Test Cases: The Osbert Oglesby Case Study 452
- 14.13** Glass-Box Unit-Testing Techniques 455
 - 14.13.1 Structural Testing: Statement, Branch, and Path Coverage 455
 - 14.13.2 Complexity Metrics 457
- 14.14** Code Walkthroughs and Inspections 458
- 14.15** Comparison of Unit-Testing Techniques 458
- 14.16** Cleanroom 459