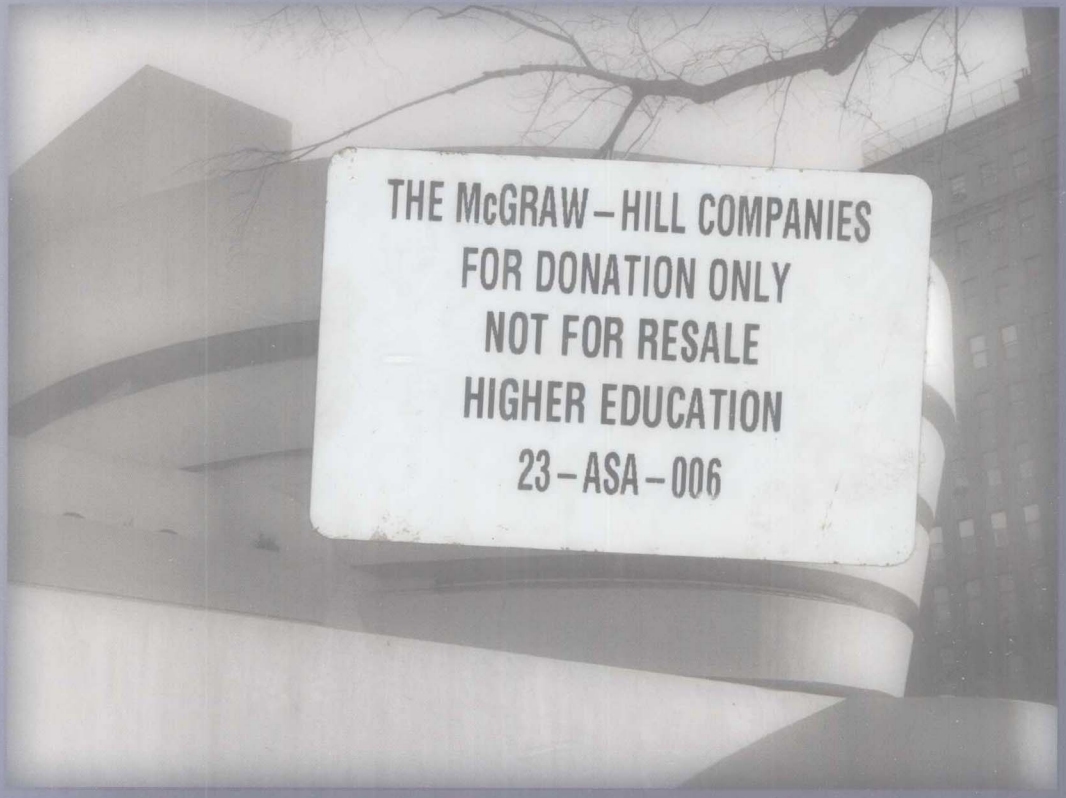
OBJECT-ORIENTED & CLASSICAL SOFTWARE ENGINEERING

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

The McGraw-Hill Companies

Mc Graw Hill **Higher Education**

OBJECT-ORIENTED AND CLASSICAL SOFTWARE ENGINEERING, SIXTH EDITION

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

www.mhhe.com

The following are registered trademarks:

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