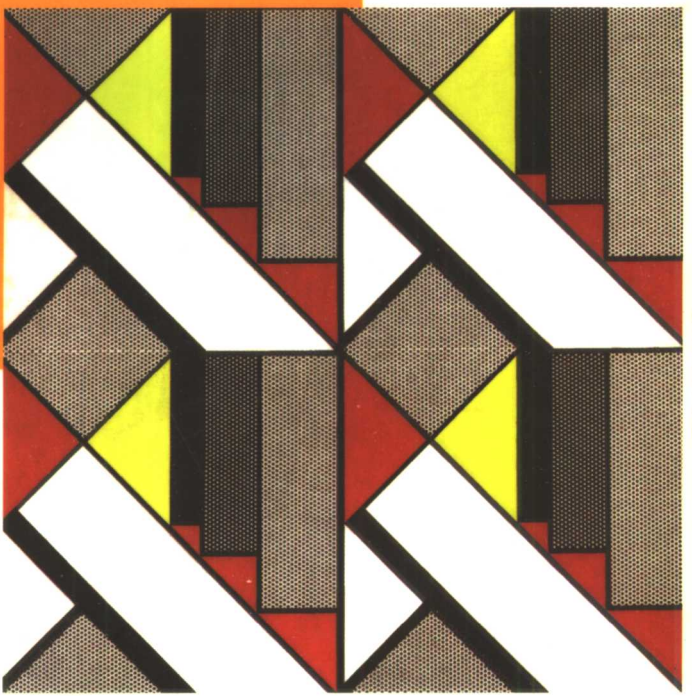
CLASSICAL AND OBJECT-ORIENTED SOFTWARE ENGINEERING

With UML and C++

STEPHEN R. SCHACH

传统和面向对象的软件工程

第4版



FOURTH EDITION

McGraw-Hill Book Co
机械工业出版社

CLASSICAL AND OBJECT-ORIENTED SOFTWARE ENGINEERING

with UML and C++

FOURTH EDITION

Stephen R. Schach
Vanderbilt University

世界图书出版公司
McGraw-Hill Book Co

WCB/McGraw-Hill

A Division of The McGraw-Hill Companies

CLASSICAL AND OBJECT-ORIENTED SOFTWARE ENGINEERING WITH UML AND C++

ISBN 0-07-290168-3

IE ISBN 0-07-116871-0

Library of Congress Cataloging-in-Publication Data

Schach, Stephen R.
Classical and object-oriented software engineering with UML and C++ / Stephen R. Schach.—4th ed.
p. cm.
First-2nd eds. published under title: Software engineering.
Third ed. published under title: Classical and object-oriented software engineering.
Includes bibliographical references and index.
ISBN 0-07-290168-3
1. Software engineering. 2. Object-oriented programming (Computer science) I. Schach, Stephen R. Software engineering.
QA76.758.S318 1998
005.1—dc21 98-15171

http://www.mhhe.com

The following are registered trademarks:

Access
ActiveX
ADF
ADW
Aide-de-Camp
Analyst/Designer
Apple
AT&T
Bachman Product Set
Battlemap
Borland
Bull
CA-Tellaplan
CCC
Coca-Cola
COM
DB2
Demo II
Emeraude
Excel
Excelerator
Ford
Foundation
FoxBASE
Guide
Hewlett-Packard
Honeywell
Hypercard
Hypertalk
IBM
IEW
IMS/360
Informix
Iona
Java
Lotus 1-2-3
Lucent Technologies
Macintosh
MacProject
Method/1
Microsoft
Motif
MS-DOS
MVS/360
Natural
Netscape
The New York Times
Newton
Nomad
Object C
ObjectBroker
Objective-C
OLE
OMTool
1-800-FLOWERS
OpenWindows
OpenDoc
Oracle
ORB Plus
ORBIX
OS/360
OS/370
OS/VS2
Post-it
PowerBuilder
Project
PVCS
QAPartner
RAMIS-II
Rational
Rose
SoftBench
Software through Pictures
Source Safe
SPARCstation
SQL
Statemate
Sun
Sun Microsystems
SunView
System Architect
Teamwork
UNIX
VAX
Visigenic
VM/370
VMS
The Wall Street Journal
Windows 95
Word
X11
XRunner

PREFACE

The title of this book, *Classical and Object-Oriented Software Engineering with UML and C++*, is somewhat surprising. After all, there is virtually unanimous agreement that the object-oriented paradigm is superior to the classical (structured) paradigm. It would seem obvious that an up-to-date software engineering textbook should describe only the object-oriented paradigm, and treat the classical paradigm at best as a historical footnote.

That is not the case. Despite the widespread enthusiasm for the object-oriented paradigm and the rapidly accumulating evidence of its superiority over the classical paradigm, it is nevertheless essential to include material on the classical paradigm. There are three reasons for this. 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.

The second reason why both the classical and object-oriented paradigms are included is that technology transfer is a slow process. The vast majority of software organizations have not yet adopted the object-oriented paradigm. It is therefore likely that many of the students who use this book will be employed by organizations that still use classical software engineering techniques. Furthermore, even if an organization is now using the object-oriented approach for developing new software, existing software still has to be maintained, and this legacy software is not object-oriented. Thus, excluding classical material would not be fair to the students who use this text.

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

The Fourth Edition differs from the Third Edition in two ways. First, many new topics are introduced in this edition. Second, the material has been rearranged to support both one- and two-semester software engineering curricula; this is described in the next section.

With regard to new topics, Unified Modeling Language (UML) permeates this edition; this is reflected in the title of the book. In addition to utilizing UML for object-oriented analysis and object-oriented design, UML has also been used wherever there are diagrams depicting objects and their interrelationships. UML has become a de facto software engineering standard and this is reflected in the Fourth Edition.

Another new topic is design patterns. This material is part of a new chapter on reuse, portability, and interoperability. Other reuse topics in this chapter include software architecture and frameworks. Underlying all the reuse material is the importance of object reuse. The portability sections include material on Java. With regard to interoperability, there are sections on topics like OLE, COM, ActiveX, and CORBA.

There is also a new chapter on planning and estimating, especially for the object-oriented paradigm. The chapter therefore includes new material on feature points and COCOMO II.

The synchronize-and-stabilize life-cycle model used by Microsoft has been included in this edition. The associated team organization method is also described.

As in the previous edition, particular attention is also paid to object-oriented life-cycle models, object-oriented analysis, object-oriented design, management implications of the object-oriented paradigm, and to the testing and maintenance of object-oriented software. Metrics for objects are also included. In addition, there are many briefer references to objects, a paragraph or even just a sentence in length. The reason is that the object-oriented paradigm is not just concerned with how the various phases are performed, but rather permeates the way we think about software engineering. Object technology pervades this book.

The software process is still the concept that underlies the book as a whole. In order 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, material on SPICE has been added to the sections on the Capability Maturity Model (CMM) and ISO 9000.

Another topic that still is stressed is CASE. I also continue to emphasize the importance of maintenance and the need for complete and correct documentation at all times.

The software process is essentially language-independent and this is again reflected in the Fourth Edition. The few code examples are in C++. However, care has been taken to make this material accessible to readers with little or no knowledge of C++ by providing explanations of constructs that are specific to C++.

With regard to prerequisites, it is assumed that the reader is familiar with one high-level programming language such as Pascal, C, C++, Ada, BASIC, COBOL, FORTRAN, or Java. In addition, the reader is expected to have taken a course in data structures.

HOW THE FOURTH EDITION IS ORGANIZED

The Third Edition of this book was written for a one-semester, project-based software engineering course. The book accordingly consisted of two parts. Part 2 covered the life cycle, phase by phase; the aim was to provide the students with the knowledge and skills needed for the Term Project. Part 1 contained the theoretical material needed to understand Part 2. For example, Part 1 introduced the reader to CASE, metrics, and testing because each chapter of Part 2 contained a section on CASE tools for that phase, a section on metrics for that phase, and a section on testing during that phase. Part 1 was kept short to enable the instructor to start Part 2 relatively early in the semester. In this way, the class could begin developing the Term Project as soon as possible. The need to keep Part 1 brief meant that I had to include topics like reuse, portability, and team organization in Part 2. Thus, while the students were working on their term projects, they learned additional theoretical material.

However, there is a new trend in software engineering curricula. More and more computer science departments are realizing that the overwhelming prepon-

derance 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 there is almost always a small project of some sort). The second course is a major team-based term project, usually a capstone project. When the Term Project is carried out in the second semester, there is no need for the instructor to rush to start Part 2.

In order to cater to both the one- and two-semester course sequences, I have rearranged the material of the previous edition and added to it. Part 1 now includes two more chapters, but the material of those two chapters is not a prerequisite for Part 2. First, Chapter 7 is entitled "Reusability, Portability, and Interoperability." The theme of this chapter is the need to develop reusable portable software that can run on a distributed heterogeneous architecture such as client-server.

Second, some instructors who adopted the Third Edition have told me that they were uncomfortable with a separate planning and estimating phase between the specification phase and the design phase. They agreed that accurate estimates of cost and duration are not possible until the specifications are complete (although sometimes we are required to produce estimates earlier in the life cycle). However, they felt that these planning and estimating activities did not merit a complete phase, particularly because they comprise only about 1 percent of the total software life cycle. Accordingly, I have dropped the separate planning phase and incorporated these activities at the end of the specifications phase. The various planning activities that are performed are described in Chapter 8, entitled "Planning and Estimating." This material, too, may be delayed in order to start Part 2. In addition to these two chapters, certain sections of other chapters (such as Section 2.12) may also be deferred and taught in parallel with Part 2. All material that can be postponed in this way is marked with ❖.

Thus, an instructor who is teaching a one-semester (or one-quarter) sequence using the Fourth Edition covers most of Chapters 1 through 6, and then starts Part 2 (Chapters 9 through 15). Chapters 7 and 8 can then be taught in parallel with Part 2. When teaching the two-semester sequence, the chapters of the book are taught in order; the class is now fully prepared for the semester-long team-based Term Project.

In order to ensure that the key software engineering techniques of Part 2 are truly 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, at the end of each chapter there is a new continuing major Case Study. A detailed solution to the Case Study is presented. The solution to each phase of the Case Study is generally too large to appear in the chapter itself. Instead, only key points of the solution are presented in the chapter, and the complete material appears at the end of the book (Appendices C through I). The rapid prototype and detailed C++ implementation are available via the World-Wide Web at http://www.mhhe.com/engcs/compsci/schach.

THE PROBLEM SETS

As in the previous edition, there are four types of problems. First, at the end of each chapter there are a number of exercises intended to highlight key points. These exercises are self-contained; the technical information for all of the exercises can be found in this book.

Second, there is a major 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 15 separate components, each tied to the relevant chapter. For example, design is the topic of Chapter 12, so in that chapter the component of the Term Project is concerned with designing the software for the project. 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 instructors may freely apply the 15 components to any other project they choose.

Because this book is written for use by graduate students as well as upperclass undergraduates, the third 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 to answer a question relating to 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 fourth type of problem relates to the Case Study. This type of problem was introduced in the Third Edition in response to instructors who told me that they believe their students learn more by modifying an existing product than by developing a product from scratch. Many senior software engineers in the industry agreed 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 than 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 readily available as described at the end of the previous section.

The *Instructor's Solution Manual*, available from McGraw-Hill, contains detailed solutions to all the exercises, as well as to the Term Project. In addition, the *Instructor's Solution Manual* contains transparency masters for all the figures in this book. The transparency masters can also be downloaded from www.mhhe.com/engcs/compsci/schach.

ACKNOWLEDGMENTS

I am indebted to those who reviewed this edition, including:

Thaddeus R. Crews, Jr., Western Kentucky University
Eduardo B. Fernandez, Florida Atlantic University
Michael Godfrey, Cornell University
Thomas B. Horton, Florida Atlantic University
Gail Kaiser, Columbia University
Laxmikant V. Kale, University of Illinois
Chung Lee, California State Polytechnic University at Pomona
Susan Mengel, Texas Tech University
David S. Rosenblum, University of California at Irvine
Shmuel Rotenstreich, George Washington University
Wendel Scarbrough, Azusa Pacific University
Gerald B. Sheble, Iowa State

I am particularly grateful to two of the reviewers. Thad Crews made many creative pedagogic suggestions. As a consequence, it is easier to teach from this book and to learn from it. Laxmikant Kale pointed out a number of weaknesses. I am grateful to him for his meticulous reading of the entire manuscript.

I should like to thank three individuals who have also made contributions to earlier books. First, Jeff Gray has once again made numerous insightful suggestions. In particular, I am grateful for his many ideas regarding Chapter 7. Also, he is once again a coauthor of the *Instructor's Solution Manual*. Second, my son David has made a number of helpful contributions to the book and is also a coauthor of the *Instructor's Solution Manual*. Third, I thank Saveen Reddy for drawing my attention to the quotation from Marcus Aurelius that appears in the last Just in Case You Wanted to Know box.

With regard to my publishers, McGraw-Hill, I am especially grateful to executive editor Betsy Jones, sponsoring editor Brad Kosirog, and project manager Paula Buschman.

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 family and my current book project. Now that my children are assisting with my books, writing has become a family activity. For the seventh time, it is my privilege to dedicate this book to my wife, Sharon, and my children, David and Lauren, with love.

Stephen R. Schach

BRIEF CONTENTS

CONTENTS

CHAPTER 7
Reusability, Portability, and Interoperability 217

CHAPTER 8
Planning and Estimating 262