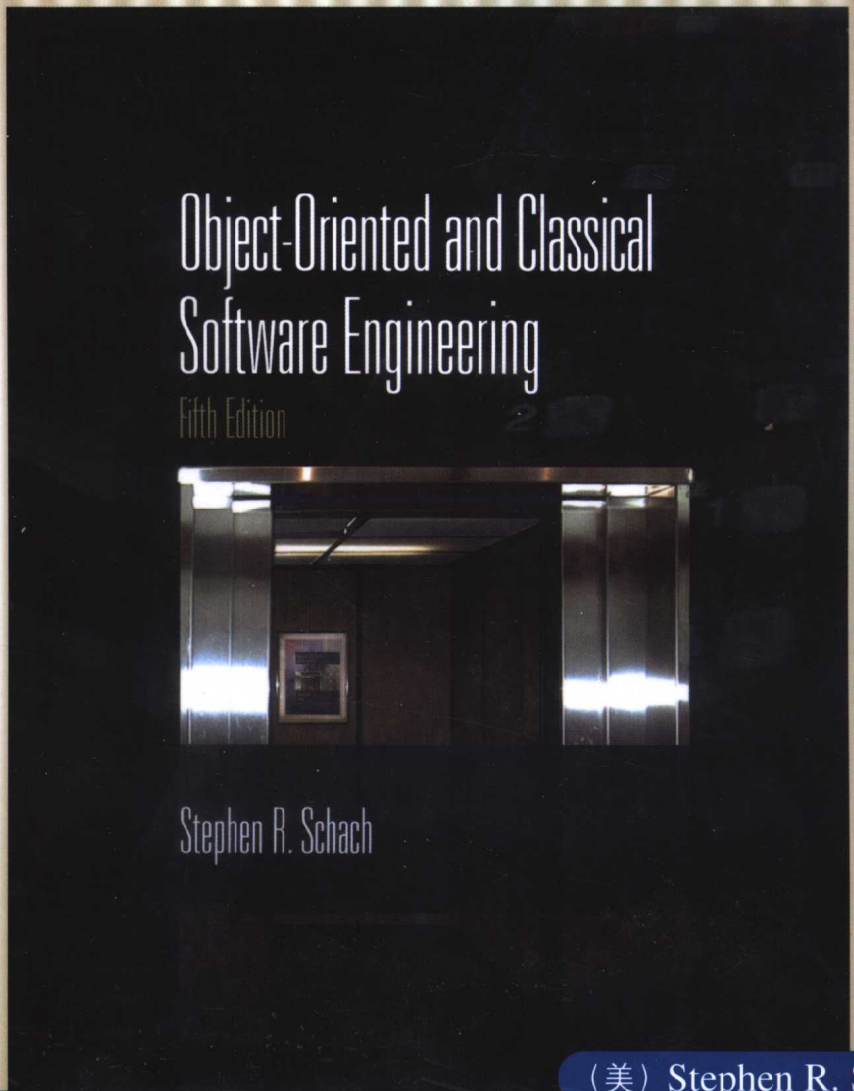
经 典 原 版 书 库

面向对象与经典软件工程

（英文版 · 第5版）



（美）Stephen R. Schach 著

机械工业出版社
China Machine Press
中信出版社
CITIC PUBLISHING HOUSE
Mc Graw Hill

经 典 原 版 书 库

面向对象与经典软件工程

（英文版 · 第5版）

Object-Oriented and Classical Software Engineering

(Fifth Edition)

（美） Stephen R. Schach 著

机械工业出版社 China Machine Press
中信出版社 CITIC PUBLISHING HOUSE

本书版权登记号：图字：01-2001-5203

图书在版编目（CIP）数据

面向对象与经典软件工程：第5版/（美）沙赫（Schach, S. R.）著. - 北京：机械工业出版社，2002.8

（经典原版书库）

书名原文：Object-Oriented and Classical Software Engineering, Fifth Edition

ISBN 7-111-10843-4

Ⅰ. 面… Ⅱ. 沙… Ⅲ. 软件工程 - 英文 Ⅳ. TP311.5

中国版本图书馆CIP数据核字（2002）第063183号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码100037）

责任编辑：华章

北京忠信诚印刷厂印刷·新华书店北京发行所发行

2002年8月第1版第1次印刷

787mm × 1092mm1/16 · 40.5印张

印数：0 001-3 000册

定价：59.00元

To Sharon, David, and Lauren

PREFACE

The fourth edition of this book was published in two versions, one with code examples presented in C++ and the other in Java. However, software engineering essentially is language independent, and in any event, there are relatively few code examples in this book. Accordingly, in this edition, I 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 utilized the pseudocode instruction *print*. (The one exception is the new case study, where complete implementation details are given in both C++ and Java.) Therefore, the fifth edition can be considered a unification of the two versions of the fourth edition.

Pedagogics is the theme of the fifth edition. All through this book, I added material to highlight key aspects of each chapter. For example, there are How to Perform boxes that summarize important techniques such as object-oriented analysis and object-oriented design. In addition, new synopses and outlines assist both the student and the instructor. Also, to provide additional material on how to perform the various techniques of software engineering, the case study in this edition is presented in greater detail than in the fourth edition.

The fourth edition included a chapter entitled "Teams and the Tools of Their Trade." As part of the stress on pedagogics in this new edition, the material has been updated and split into two, to focus more clearly on each of the separate underlying topics. In this edition, Chapter 4 is devoted to teams, whereas the tools used by software engineers are described in Chapter 5.

As before, I include both classical and object-oriented material, notwithstanding the virtually unanimous agreement that the object-oriented paradigm is superior to the classical (structured) paradigm. My decision might surprise some readers; surely 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.

This 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 nevertheless is 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 therefore is 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 using 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 not be fair 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 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.

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. In the intervening three years, UML has been formally standardized and become so widely used that any textbook that does not use UML to

describe object-oriented analysis and design immediately would be obsolete. Therefore, I continue to use UML for object-oriented analysis and object-oriented design, as well as wherever diagrams depict objects and their interrelationships.

Another then-new topic introduced into the fourth edition was design patterns. As with UML, design patterns now are part of mainstream software engineering. The material on design patterns therefore has been retained and strengthened.

A new topic in this edition is extreme programming (XP). XP still is controversial, but I feel that students need an overview of the topic so they can decide for themselves whether XP is merely a fad or a genuine major breakthrough in software engineering.

In the previous edition, 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 techniques unless they appreciate the importance of the basics of software engineering.

As in the fourth 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 also are included. In addition, there are many briefer references to objects, a paragraph or even only 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 still is 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) and ISO/IEC 15504 (SPICE) has been updated, and material on ISO/IEC 12207 has been added.

As in the fourth 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 that 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 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 Fifth Edition Is Organized

Like the fourth edition of this book, the fifth 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 life cycle, phase by phase, 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 phase, a section on metrics for that phase, and a section on testing during that phase. 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 and taught in parallel with Part 2. The class then 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 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 consists of 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 fifth 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 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 new case study is presented toward the end of each chapter. This detailed solution provides the second illustration of each technique.

THE PROBLEM SETS

As in the previous edition, there are four types of problems. First, 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.

Second, 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 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 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 fourth type of problem relates to the case study. This type of problem was first introduced in the third edition in response to instructors who feel that their students learn more by modifying an existing product than by developing a 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 web site also has transparency masters for all the figures in this book, as well as 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 am indebted to those who reviewed this edition, including:

Arvin Agah (University of Kansas)
Thaddeus R. Crews, Jr. (Western Kentucky University)
Eduardo B. Fernandez (Florida Atlantic University)
Michael Godfrey (Cornell University)
Scott Hawker (University of Alabama)
Thomas B. Horton (Florida Atlantic University)
Gail Kaiser (Columbia University)
Laxmikant V. Kale (University of Illinois)
Helene Kershner (University of Buffalo)
Chung Lee (California State Polytechnic University at Pomona)

Richard A. Lejk (University of North Carolina, Charlotte)
Susan A. Mengel (Texas Technological University)
David S. Rosenblum (University of California at Irvine)
Shmuel Rotenstreich (George Washington University)
Wendel Scarbrough (Azusa Pacific University)
Gerald B. Sheble (Iowa State)
Jie We (City University of New York)
David Workman (University of Central Florida)

I thank two individuals who made contributions to earlier books. First, Jeff Gray once again made numerous insightful suggestions. In particular, I am grateful for his many ideas regarding Chapter 8. Also, he once again is a coauthor of the *Instructor's Solution Manual.* Second, my son David has made a number of helpful contributions to the book and again is a coauthor of the *Instructor's Solution Manual.*

Since 1999, I have been involved in joint research with Dr. Amir Tomer of RAFAEL and the Technion, Israel Institute of Technology. The papers we wrote together are nominally on maintenance. However, the issue underlying our research is the nature of software engineering. A direct consequence of working with Amir is that I gained new insight into software engineering. I have incorporated many of these ideas into this edition.

Turning now to my publisher, McGraw-Hill, I am truly grateful to executive editor Betsy Jones and developmental editor Emily Gray for their assistance from start to finish. I particularly appreciate their suggestions regarding giving equal stress to both C++ and Java in an integrated volume. Rick Hecker was the ideal project manager in every way. I was most fortunate to have Gnomi Schrift Gouldin as the copy editor for this book. She greatly improved the readability of my manuscript, and I am grateful for her many suggestions.

I would like to thank the many instructors from all over the world who sent me e-mail concerning the fourth 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. First, I thank my students at Vanderbilt for their many questions and comments, both inside and outside the classroom. I also an most grateful for the provocative questions and constructive suggestions e-mailed me by students from all over the world. 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 tenth 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