Preface

The sixth edition of this book, published in 2004, was vastly different to the fifth edition. More than half of the sixth edition consisted of new material, much of it on the Unified Process, and the topics that were retained from the fifth edition were rewritten from a more modern viewpoint.

Many instructors were kind enough to provide feedback on the sixth edition. They approved of the drastic changes I had made in the sixth edition, but suggested that the aim of the seventh edition should be to update the sixth edition, rather than once again make widespread changes. In addition, a number of instructors have asked for a new running case study in Part 2 of the book.

Accordingly, there are two types of changes in the seventh edition:

- The book has been updated throughout. In particular, I have considerably expanded the material on agile processes and open-source software development in Chapters 2 and 4. The references have been updated, and there are new problems in every chapter.
- I have replaced the Osbert Oglesby case study, used to illustrate the techniques of software development in Chapters 10 through 15, with the MSG Foundation case study.

Features Retained from the Sixth Edition

- The Unified Process is still largely the methodology of choice for object-oriented software development. Throughout this book, the student is therefore exposed to both the theory and the practice of the Unified Process.
- In Chapter 1, the strengths of the object-oriented paradigm are analyzed in depth.
- The iterative-and-incremental life-cycle model has been introduced as early as possible, namely, in Chapter 2. Furthermore, 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 processes of the Unified Process are introduced, and the need for two-dimensional life-cycle models is explained.
- A wide variety of ways of organizing software teams are presented in Chapter 4 ("Teams"), including teams for agile processes and for open-source software development.
- Chapter 5 ("The Tools of the Trade") includes information on important classes of CASE tools.
- The importance of continual testing is stressed in Chapter 6 ("Testing").
- Objects continue to be the focus of attention in Chapter 7 ("From Modules to Objects").
- The material on interoperability, which was removed from Chapter 8 ("Reusability and Portability") in the sixth edition, has not been replaced. 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.

- The new IEEE standard for software project management plans is again presented in Chapter 9 (“Planning and Estimating”).
- Chapter 10 (“Requirements”), Chapter 12 (“Object-Oriented Analysis”), and Chapter 13 (“Design”) are largely devoted to the workflows (activities) of the Unified Process. For obvious reasons, Chapter 11 (“Classical Analysis”) has not been changed, other than the case study.
- The material in Chapter 14 (“Implementation”) clearly distinguishes between implementation and integration.
- The importance of postdelivery maintenance is stressed in Chapter 15.
- Chapter 16 provides additional material on UML to prepare the student 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.
- As before, there are two running case studies. The MSG Foundation case study and the elevator problem case study have been developed using the Unified Process. As usual, Java and C++ implementations are available online at www.mhhe.com/schach.
- 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 postdelivery maintenance.
- In all the previous editions, I have 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 sixth 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, 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 rather permeates the way we think about software engineering. Object technology again 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), 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.)

- As in the sixth edition, this book contains over 600 references. I have 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

There is now almost unanimous agreement that the object-oriented paradigm is superior to the classical paradigm. Nevertheless, I feel that attempting to eliminate all mention of the classical paradigm is 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, all through 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 Seventh Edition Is Organized

Like the sixth edition of this book, the seventh edition is written for both the traditional one-semester and the newer two-semester software engineering curriculum, now growing in popularity. 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, and 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 often includes a small project of some sort). The second course comprises a major team-based term project. This is 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 seventh edition covers most of Chapters 1 through 7 and then starts Part 2 (Chapters 10 through 16). Chapters 8 and 9 can 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 MSG Foundation case study is presented. This detailed solution provides the second illustration of each technique.

The Problem Sets

As in the previous edition, this book has five types of problems. First, there are 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 15 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 15 components to any other project that he or she chooses.

Because this book has been 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 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 fifth type of problem relates to the 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 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/schach.

The website also has material for instructors, including a complete set of PowerPoint lecture notes and detailed solutions to all the exercises as well as to the term project.

Material on UML

This book makes substantial use of the Unified Modeling Language (UML). If the students do not have previous knowledge of UML, this material may be taught in two ways. I prefer to teach UML on a just-in-time basis; that is, each UML concept is introduced just before it is needed. The following table describes where the UML constructs used in this book are introduced.

Construct	**Section in Which the Corresponding UML Diagram Is Introduced**
Class diagram, note, inheritance (generalization), aggregation, association, navigation triangle	Section 7.7
Use case	Section 10.4.3
Use-case diagram, use-case description	Section 10.7
Stereotype	Section 12.1
Statechart	Section 12.6
Interaction diagram (sequence diagram, collaboration diagram)	Section 12.15

Alternatively, Chapter 16 contains an introduction to UML, including material above and beyond what is needed for this book. Chapter 16 may be taught at any time; it does not depend on material in the first 15 chapters. The topics covered in Chapter 16 are as given in the following table.

Construct	Section in Which the Corresponding UML Diagram Is Introduced
Class diagram, aggregation, multiplicity, composition, generalization, association	Section 16.2
Note	Section 16.3
Use-case diagram	Section 16.4
Stereotype	Section 16.5
Interaction diagram	Section 16.6
Statechart	Section 16.7
Activity diagram	Section 16.8
Package	Section 16.9
Component diagram	Section 16.10
Deployment diagram	Section 16.11

Acknowledgments

I greatly appreciate the constructive criticisms and many helpful suggestions of the reviewers of the six 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

Michael Buckley
State University New York, Buffalo

Catherine Lowry Campbell
New Jersey Institute of Technology

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

Frances Grodzinsky
Sacred Heart University

Jim Han
Florida Atlantic 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
Werner Krandick
Drexel University
Owen Lavin
DePaul University
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
Donald Needham
United States Naval Academy
Ron New
Johns Hopkins University
David Notkin
University of Washington
Andy Podgurski
Case Western Reserve University
Hal Render
University of Colorado, Colorado Springs
David C. Rine
George Mason University
David S. Rosenblum
University of California, Irvine
Shmuel Rotenstreich
George Washington University
Mansur Samadzadeh
Oklahoma State University
John H. Sayler
University of Michigan
Wendel Scarborough
Azusa Pacific University
Bob Schuerman
State College, Pennsylvania
Gerald B. Sheble
Iowa State
Fred Strauss
Polytechnic University
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

Robert M. Cubert
University of Florida
Michael Hoffman
California State University, Long Beach
Saeed S. Monemi
California State Polytechnic University, Pomona
Linda Ott
Michigan Technological University

James Purtilo
University of Maryland
Steven Shaffer
Pennsylvania State University
Fred Strauss
Polytechnic University

I would like to thank the numerous instructors from all over the world who sent me e-mail regarding the sixth edition. As always, I am exceedingly appreciative of their suggestions, comments, and criticisms. In particular, I thank Professor Michael Haugrud (Minnesota State University Moorhead) for suggesting improvements to the Winburg mini case study. I look forward with anticipation to receiving instructors' feedback on this edition also. My e-mail address is srs@vuse.vanderbilt.edu.

Students, too, continue to be most helpful. Once more I thank my students at Vanderbilt University for their provocative questions and constructive suggestions, both inside and outside the classroom. I also am most appreciative of the questions and comments on the sixth edition e-mailed to me by students from all over the world. Special thanks go to Benjamin Polak, a student at the University of Erlangen, Germany, for suggesting improvements to the elevator problem case study. As with the previous editions, I look forward keenly to student feedback on this edition, too.

I warmly thank three individuals who have also made significant contributions to previous editions of this book. First, Kris Irwin once again provided a complete solution to the term project, including implementing it in both Java and C++. Second, Jeff Gray implemented the MSG Foundation case study. Third, Lauren Ryder was again a co-author of the *Instructor's Solution Manual* and contributor to the PowerPoint slides.

I turn now to McGraw-Hill. My publisher Alan Apt and my developmental editor Rebecca Olson provided assistance and guidance throughout the project. I thank copyeditor Lucy Mullins for her many helpful suggestions. It was a pleasure to work with marketing manager Michael Weitz, media manager Christina Nelson, production supervisor Kara Kudronowicz, and proofreader Carrie Barker. The striking cover of this book was designed by Michelle Whitaker and Christopher Reese, both of whom I warmly thank. Finally, I particularly wish to thank production manager Lora Kalb for her endless help and for keeping the project on schedule. She invariably went the extra mile—and more.

As always, I enjoyed working with the highly competent compositors at Interactive Composition Corporation. I also thank ICC project manager Deepti Hingle.

Finally, as always, I thank my wife, Sharon, for her continual support. As with all my previous books, I did my utmost to try to ensure that family commitments took precedence over writing. However, when deadlines loomed, this was sometimes not possible. At such times, she was always understanding, and for this I am most grateful.

It is my privilege to dedicate my thirteenth book to my grandson, Jackson, with love.

Stephen R. Schach