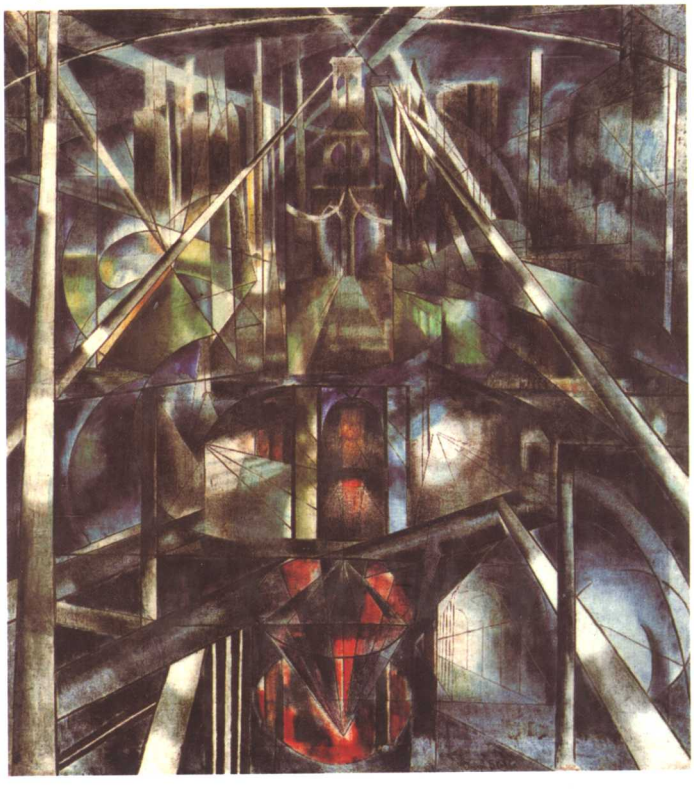

STEPHEN R. SCHACH

Third Edition

CLASSICAL and OBJECT-ORIENTED SOFTWARE ENGINEERING

CLASSICAL AND OBJECT-ORIENTED SOFTWARE ENGINEERING

THIRD EDITION

Stephen R. Schach
Vanderbilt University

Irwin McGraw-Hill

Boston, Massachusetts Burr Ridge, Illinois Dubuque, Iowa
Madison, Wisconsin New York, New York San Francisco, California St. Louis, Missouri

Irwin/McGraw-Hill

A Division of The McGraw-Hill Companies

Earlier editions titled *Software Engineering*

Senior sponsoring editor: Elizabeth A. Jones
Marketing manager: Brian Kibby
Project editor: Rebecca Dodson
Production supervisor: Dina L. Treadaway
Compositor: Weimer Graphics, Inc.
Typeface: 10/12 Times Roman
Printer: R. R. Donnelley & Sons Company
Cover illustrator: Joseph Stella (1877–1946), *Brooklyn Bridge,* oil on canvas, 84″ x 76″, Yale University Art Gallery, Collection of Societe Anonyme

Library of Congress Cataloging-in-Publication Data

Schach, Stephen R.
Classical and object-oriented software engineering / Stephen R. Schach. — 3rd ed.
p. cm.
Previous eds. published under title: Software engineering.
Includes bibliographical references and indexes.
ISBN 0-256-18298-1
1. Software engineering. 2. Object-oriented programming (Computer science) I. Schach, Stephen R. Software engineering. II. Title.
QA76.758.S33 1996
005.1—dc20 94–46905

Printed in the United States of America
5 6 7 8 9 0 DO 9 8 7

PREFACE

The Second Edition of *Software Engineering* was published in 1993. At that time there were two major approaches to software development, namely the structured paradigm and the object-oriented paradigm. The structured paradigm was a tried and trusted approach, but it was not always successful. On the other hand, the object-oriented paradigm seemed promising, but no more than that. The Second Edition reflected this attitude. The book certainly included material on objects and on object-oriented design, but at that time it was premature to stress a new paradigm that had not been proven to be superior to the structured paradigm.

In the 3 years since the Second Edition was published, evidence has been steadily mounting that the object-oriented paradigm is superior to classical software engineering approaches. In fact, a textbook exclusively devoted to object-oriented software engineering would now be justified.

If that is so, then why is this book entitled *Classical and Object-Oriented Software Engineering?* Why are the classical techniques even mentioned? There are two reasons for this.

First, this book is a textbook at the senior undergraduate or first year graduate level, and it is likely that many 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 existing software is not object-oriented. Thus, excluding classical material would not be fair to students using this text.

The second reason why both classical and object-oriented techniques are included is that it is impossible to understand why object-oriented technology is superior to classical technology without fully understanding classical approaches and how they differ from the object-oriented approach. Thus, the classical and object-oriented approaches are not merely both described in this book, they are compared, contrasted, and analyzed. This ensures that the reader will fully appreciate why so many software professionals feel that the object-oriented approach is superior to classical approaches. Furthermore, if the student is employed at an organization that has not yet adopted object-oriented technology, he or she will be able to advise that organization regarding both the strengths and the weaknesses of the new paradigm.

Thus, the major change in this edition is that the object-oriented paradigm is emphasized. Objects are introduced in the very first chapter and are discussed throughout the book. Chapter 6, entitled "Introduction to Objects," provides clear definitions of basic object-oriented concepts such as classes, objects, inheritance, polymorphism, and dynamic binding (the chapter is an extended version of Chapter 9 of the second edition). There is a new chapter on object-oriented analysis, a topic that was not covered in the second edition. Particular attention is also paid to object-oriented life-cycle models, object-oriented design, management implications of the object-oriented paradigm, and to the testing and maintenance of object-oriented software. Metrics for objects and cohesion and coupling of objects are also covered. In addition, there are many briefer references to objects, usually only a paragraph or even 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. As a result, object-oriented 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 stress on metrics is maintained.

The third edition continues and extends other themes of the previous editions. For instance, the second edition contained a discussion of the Capability Maturity Model (CMM) and how it was being used to improve the software process and thereby boost productivity. In this edition, the ISO 9000-series is also discussed and is contrasted with the CMM.

There have been a number of developments within the area of Computer-Aided Software Engineering (CASE). On the one hand, some organizations have become disillusioned with CASE, whereas others have introduced CASE and have observed a marked improvement in areas such as productivity, software quality, and employee morale. This book gives a balanced view of CASE and explains why organizations have had such differing experiences with it. CASE tools for the object-oriented paradigm are also included.

Topics that continue to be emphasized throughout the book include the importance of maintenance and the need for complete and correct documentation at all times. The importance of software reuse is still stressed, but now within the context of objects.

The book is still essentially language-independent. The few code examples are in C++. To be more precise, wherever possible the "C subset of C++" has been used. In addition, care has been taken to use as few C idioms as possible so that the material can also be understood by readers with little or no knowledge of C. The only chapter where C++ (rather than C) is employed is Chapter 6, and detailed explanations of specific C++ constructs have been provided there. In addition, the implementation of the Case Study in Appendix I uses some C++ constructs.

With regard to prerequisites, it is assumed that the reader is familiar with one high-level programming language such as Pascal, C, BASIC, COBOL, or FORTRAN. Although most of the examples are in C, no previous knowledge of C is needed. In addition, the reader is expected to have taken a course in data structures.

How the Third Edition Is Organized

The order of the chapters reflects the order of the phases of the software life cycle. Specifically, Part Two of this book (Chapters 7 through 14) consists of a phase-by-phase treatment of the software life cycle, starting with the requirements phase and ending with the maintenance phase. In order to prepare the reader for this material, Part One contains the background material needed to understand the second part of the book. For example, Part One introduces the reader to CASE, metrics, and testing because each chapter of Part Two contains a section on CASE tools for that phase, a section on metrics for that phase, and a section on testing during that phase.

In order to ensure that the key software engineering techniques of Part Two 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 continuing major Case Study. A detailed solution to the Case Study is presented. The material for 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 itself and the complete material appears at the end of the book (Appendices C through I).

The Problem Sets

In this edition, there are four types of exercises. First, as before, 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 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 14 separate components, each tied to the relevant chapter. For example, design is the topic of Chapter 11, 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 14 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.

New to this edition is the fourth type of problem, namely, problems related to the Case Study. A number of instructors have 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 with whom I have discussed the issue 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 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 object-oriented analysis in a different order. In order to make it easy to modify the source code of the Case Study (Appendices C and I), the source code is available by anonymous ftp from ftp.vuse.vanderbilt.edu (129.59.100.10) in directory /pub/Software_Eng/Third_Edition, or on a diskette from Richard D. Irwin, 1333 Burr Ridge Parkway, Burr Ridge, Illinois 60521.

The Instructor's Manual contains detailed solutions to all the exercises, as well as to the term project. The Instructor's Manual is also available from Richard D. Irwin, and so are transparency masters for all the figures in this book.

Acknowledgments

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

Dan Berry
The Technion

Doug Bickerstaff
Eastern Washington University

Richard J. Botting
California State University–San Bernardino

Buster Dunsmore
Purdue University

E.B. Fernandez
Florida Atlantic University

Donald Gotterbarn
East Tennessee State University

Greg Jones
Utah State University

Peter E. Jones
University of Western Australia–Nedlands, Perth

David Notkin
University of Washington

Hal Render
University of Colorado–Colorado Springs

Bob Schuerman
State College, Pennsylvania

K.C. Tai
North Carolina State University

Laurie Werth
University of Texas–Austin

Lee White
Case Western Reserve University

George W. Zobrist
University of Missouri–Rolla

Jeff Gray, co-author of the Instructor's Manual for this edition, has made many helpful suggestions. In particular, I thank him for his ideas regarding the Z specification of Section 8.8.1. I am grateful to Saveen Reddy for his comments on Sections 6.4 through 6.6. I should also like to thank Keith Pierce, University of Minnesota, Duluth, for his helpful suggestions regarding test plans. Some of the material of the MSG Case Study, presented at the end of Chapters 7 through 13 and in Appendices C

through I, has been taken from the Term Project in the Second Edition of this book and from the Instructor's Manual for the Second Edition (co-authored by Santhosh R. Sastry).

I should like to single out three individuals at Richard D. Irwin to whom I am especially grateful. I thank senior sponsoring editor Betsy Jones, project editor Becky Dodson, and copy editor June Waldman for their many valuable contributions to this book.

Finally, I thank my family for their wholehearted support and encouragement throughout the writing of this edition. As with all my previous books, I have done my utmost to ensure that family commitments took precedence over writing. However, when deadlines loomed, this was sometimes not possible. At such times, they were always understanding, and for this I am most grateful. As always, I dedicate this book to my wife, Sharon, and my children, David and Lauren, with love.

Stephen R. Schach

BRIEF CONTENTS

CONTENTS