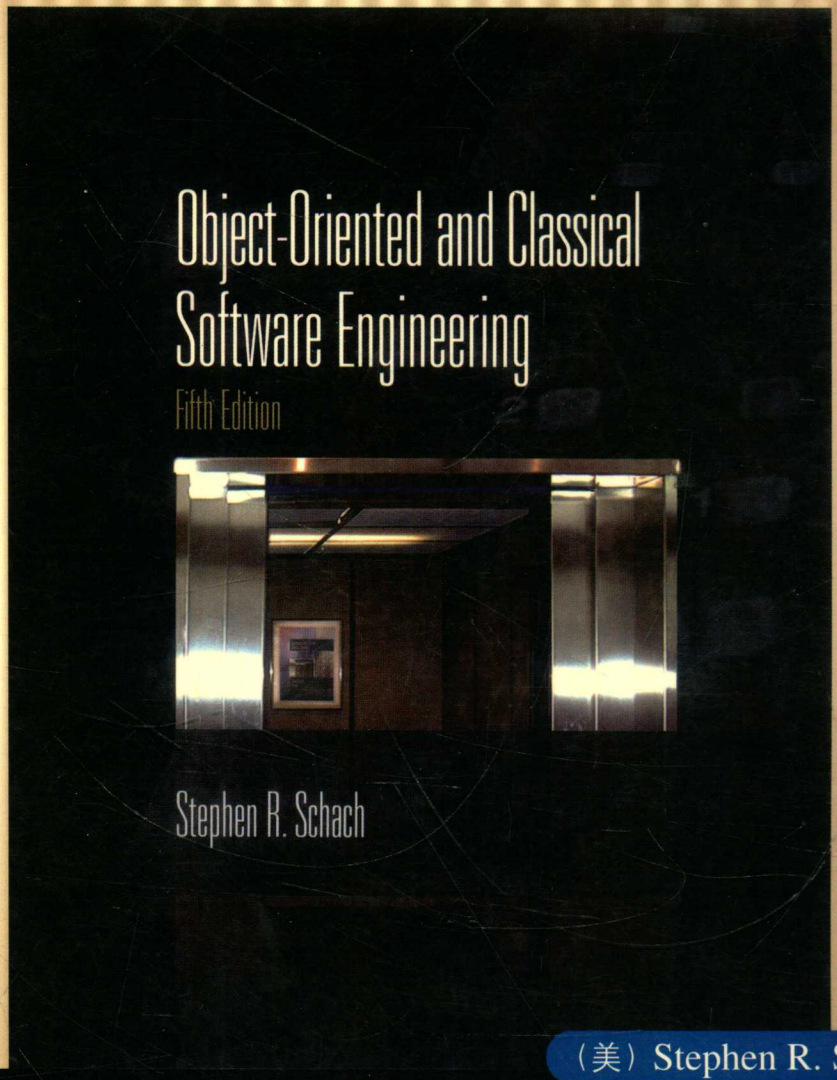
经 典 原 版 书 库

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

Stephen R. Schach: Object-Oriented and Classical Software Engineering, Fifth Edition(ISBN 0-07-239559-1).

Copyright ©2002 by The McGraw-Hill Companies,Inc.All rights reserved.

Jointly published by China Machine Press and CITIC Publishing House/McGraw-Hill.This edition may be sold in the People's Republic of China only.This book cannot be re-exported and is not for sale outside the People's Republic of China.

本书英文影印版由美国McGraw-Hill公司授权机械工业出版社和中信出版社在中国大陆境内独家出版发行，未经出版者许可，不得以任何方式抄袭、复制或节录本书中的任何部分。

版权所有，侵权必究。

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

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

To Sharon, David, and Lauren

Brief Contents

Preface xv

PART 1
Introduction to Software Engineering 1

Chapter 1
The Scope of Software Engineering 3

Chapter 2
The Software Process 30

Chapter 3
Software Life-Cycle Models 64

Chapter 4
Teams 90

Chapter 5
The Tools of the Trade 106

Chapter 6
Testing 136

Chapter 7
From Modules to Objects 167

Chapter 8
Reusability, Portability, and Interoperability 212

Chapter 9
Planning and Estimating 257

PART 2
The Phases of the Software Life Cycle 289

Chapter 10
Requirements Phase 290

Chapter 11
Specification Phase 319

Chapter 12
Objected-Oriented Analysis Phase 366

Chapter 13
Design Phase 395

Chapter 14
Implementation Phase 434

Chapter 15
Implementation and Integration Phase 474

Chapter 16
Maintenance Phase 493

试读结束：需要全本请在线购买：www.ertongbook.com

Appendix A
Broadlands Area Children's Hospital 513

Appendix B
Software Engineering Resources 518

Appendix C
Air Gourmet Case Study: C Rapid Prototype 520

Appendix D
Air Gourmet Case Study: Java Rapid Prototype 521

Appendix E
Air Gourmet Case Study: Structured Systems Analysis 522

Appendix F
Air Gourmet Case Study: Software Project Management Plan 529

Appendix G
Air Gourmet Case Study: Object-Oriented Analysis 534

Appendix H
Air Gourmet Case Study: Design for C++ Implementation 535

Appendix I
Air Gourmet Case Study: Design for Java Implementation 560

Appendix J
Air Gourmet Case Study: Black-Box Test Cases 582

Appendix K
Air Gourmet Case Study: C++ Source Code 588

Appendix L
Air Gourmet Case Study: Java Source Code 589

Bibliography 590

Author Index 617

Subject Index 623

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

试读结束：需要全本请在线购买： www.ertongbook.com

CONTENTS

Preface xv

PART 1
Introduction to Software Engineering 1

Chapter 1
The Scope of Software Engineering 3

1.1 Historical Aspects 4
1.2 Economic Aspects 7
1.3 Maintenance Aspects 8
1.4 Specification and Design Aspects 13
1.5 Team Programming Aspects 15
1.6 The Object-Oriented Paradigm 17
1.7 Terminology 21
Chapter Review 23
For Further Reading 24
Problems 25
References 26

Chapter 2
The Software Process 30

2.1 Client, Developer, and User 32
2.2 Requirements Phase 33
2.2.1 Requirements Phase Testing 34
2.2.2 Requirements Phase Documentation 35
2.3 Specification Phase 35
2.3.1 Specification Phase Testing 37
2.3.2 Specification Phase Documentation 38
2.4 Design Phase 38
2.4.1 Design Phase Testing 39
2.4.2 Design Phase Documentation 40
2.5 Implementation Phase 40
2.5.1 Implementation Phase Testing 40
2.5.2 Implementation Phase Documentation 40
2.6 Integration Phase 41
2.6.1 Integration Phase Testing 41
2.6.2 Integration Phase Documentation 42
2.7 Maintenance Phase 42
2.7.1 Maintenance Phase Testing 43
2.7.2 Maintenance Phase Documentation 43
2.8 Retirement 43
2.9 Problems with Software Production: Essence and Accidents 44
2.9.1 Complexity 45
2.9.2 Conformity 47
2.9.3 Changeability 48
2.9.4 Invisibility 49
2.9.5 No Silver Bullet? 50
2.10 Improving the Software Process 51
2.11 Capability Maturity Models 51
2.12 Other Software Process Improvement Initiatives 54
2.13 Costs and Benefits of Software Process Improvement 55
Chapter Review 57
For Further Reading 58
Problems 59
References 60

Chapter 3
Software Life-Cycle Models 64

3.1 Build-and-Fix Model 64
3.2 Waterfall Model 65
3.2.1 Analysis of the Waterfall Model 68

3.3 Rapid Prototyping Model 70
3.3.1 Integrating the Waterfall and Rapid Prototyping Models 71
3.4 Incremental Model 72
3.4.1 Analysis of the Incremental Model 73
3.5 Extreme Programming 75
3.6 Synchronize-and-Stabilize Model 77
3.7 Spiral Model 78
3.7.1 Analysis of the Spiral Model 82
3.8 Object-Oriented Life-Cycle Models 82
3.9 Comparison of Life-Cycle Models 84
Chapter Review 86
For Further Reading 86
Problems 87
References 87

Chapter 4
Teams 90

4.1 Team Organization 90
4.2 Democratic Team Approach 92
4.2.1 Analysis of the Democratic Team Approach 93
4.3 Classical Chief Programmer Team Approach 93
4.3.1 The *New York Times* Project 95
4.3.2 Impracticality of the Classical Chief Programmer Team Approach 96
4.4 Beyond Chief Programmer and Democratic Teams 97
4.5 Synchronize-and-Stabilize Teams 101
4.6 Extreme Programming Teams 102
Chapter Review 103
For Further Reading 104
Problems 104
References 105

Chapter 5
The Tools of the Trade 106

5.1 Stepwise Refinement 106
5.1.1 Stepwise Refinement Example 107
5.2 Cost–Benefit Analysis 113
5.3 Software Metrics 114
5.4 CASE 115
5.5 Taxonomy of CASE 116
5.6 Scope of CASE 118
5.7 Software Versions 122
5.7.1 Revisions 122
5.7.2 Variations 123
5.8 Configuration Control 124
5.8.1 Configuration Control during Product Maintenance 126
5.8.2 Baselines 127
5.8.3 Configuration Control during Product Development 127
5.9 Build Tools 128
5.10 Productivity Gains with CASE Technology 129
Chapter Review 131
For Further Reading 131
Problems 132
References 133

Chapter 6
Testing 136

6.1 Quality Issues 137
6.1.1 Software Quality Assurance 137
6.1.2 Managerial Independence 138
6.2 Nonexecution-Based Testing 139
6.2.1 Walkthroughs 139
6.2.2 Managing Walkthroughs 140
6.2.3 Inspections 141
6.2.4 Comparison of Inspections and Walkthroughs 143
6.2.5 Strengths and Weaknesses of Reviews 144
6.2.6 Metrics for Inspections 144
6.3 Execution-Based Testing 145
6.4 What Should Be Tested? 145
6.4.1 Utility 146
6.4.2 Reliability 147
6.4.3 Robustness 147
6.4.4 Performance 148
6.4.5 Correctness 149
6.5 Testing versus Correctness Proofs 151
6.5.1 Example of a Correctness Proof 151
6.5.2 Correctness Proof Case Study 154

6.5.3 Correctness Proof and Software Engineering 155
6.6 Who Should Perform Execution-Based Testing? 158
6.7 When Testing Stops 160
Chapter Review 160
For Further Reading 161
Problems 162
References 164

Chapter 7
From Modules to Objects 167

7.1 What Is a Module? 167
7.2 Cohesion 171
7.2.1 Coincidental Cohesion 171
7.2.2 Logical Cohesion 172
7.2.3 Temporal Cohesion 173
7.2.4 Procedural Cohesion 174
7.2.5 Communicational Cohesion 174
7.2.6 Functional Cohesion 175
7.2.7 Informational Cohesion 175
7.2.8 Cohesion Example 176
7.3 Coupling 177
7.3.1 Content Coupling 178
7.3.2 Common Coupling 178
7.3.3 Control Coupling 180
7.3.4 Stamp Coupling 180
7.3.5 Data Coupling 182
7.3.6 Coupling Example 182
7.3.7 The Importance of Coupling 182
7.4 Data Encapsulation 184
7.4.1 Data Encapsulation and Product Development 186
7.4.2 Data Encapsulation and Product Maintenance 188
7.5 Abstract Data Types 194
7.6 Infromation Hiding 195
7.7 Objects 198
7.8 Inheritance, Polymorphism, and Dynamic Binding 201
7.9 Cohesion and Coupling of Objects 203
7.10 The Object-Oriented Paradigm 204
Chapter Review 207
For Further Reading 207
Problems 208
References 209

Chapter 8
Reusability, Portability, and Interoperability 212

8.1 Reuse Concepts 212
8.2 Impediments to Reuse 214
8.3 Reuse Case Studies 216
8.3.1 Raytheon Missile Systems Division 216
8.3.2 Toshiba Software Factory 217
8.3.3 NASA Software 218
8.3.4 GTE Data Services 219
8.3.5 Hewlett-Packard 220
8.3.6 European Space Agency 221
8.4 Objects and Reuse 222
8.5 Reuse during the Design and Implementation Phases 222
8.5.1 Design Reuse 222
8.5.2 Application Frameworks 224
8.5.3 Design Patterns 225
8.5.4 Software Architecture 229
8.6 Reuse and Maintenance 230
8.7 Portability 231
8.7.1 Hardware Incompatibilities 232
8.7.2 Operating Systems Incompatibilities 233
8.7.3 Numerical Software Incompatibilities 233
8.7.4 Compiler Incompatibilities 235
8.8 Why Portability? 239
8.9 Techniques for Achieving Portability 240
8.9.1 Portable System Software 240
8.9.2 Portable Application Software 241
8.9.3 Portable Data 242
8.10 Interoperability 243
8.10.1 COM 243
8.10.2 CORBA 244
8.10.3 Comparing COM and CORBA 245
8.11 Future Trends in Interoperability 245
Chapter Review 246
For Further Reading 247
Problems 248
References 250

Chapter 9
Planning and Estimating 257

9.1 Planning and the Software Process 257
9.2 Estimating Duration and Cost 259
9.2.1 Metrics for the Size of a Product 260
9.2.2 Techniques of Cost Estimation 264
9.2.3 Intermediate COCOMO 267
9.2.4 COCOMO II 270
9.2.5 Tracking Duration and Cost Estimates 272
9.3 Components of a Software Project Management Plan 272
9.4 Software Project Management Plan Framework 274
9.5 IEEE Software Project Management Plan 274
9.6 Planning Testing 278
9.7 Planning Object-Oriented Projects 279
9.8 Training Requirements 280
9.9 Documentation Standards 281
9.10 CASE Tools for Planning and Estimating 282
9.11 Testing the Software Project Management Plan 282
Chapter Review 283
For Further Reading 283
Problems 284
References 285

PART 2
The Phases of the Software Life Cycle 289

Chapter 10
Requirements Phase 290

10.1 Requirements Elicitation 291
10.1.1 Interviews 291
10.1.2 Scenarios 292
10.1.3 Other Requirements Elicitation Techniques 293
10.2 Requirements Analysis 294
10.3 Rapid Prototyping 294
10.4 Human Factors 296
10.5 Rapid Prototyping as a Specification Technique 298
10.6 Reusing the Rapid Prototype 300
10.7 Management Implications of the Rapid Prototyping Model 302
10.8 Experiences with Rapid Prototyping 304
10.9 Techniques for Requirements Elicitation and Analysis 305
10.10 Testing during the Requirements Phase 305
10.11 CASE Tools for the Requirements Phase 306
10.12 Metrics for the Requirements Phase 307
10.13 Object-Oriented Requirements? 308
10.14 Air Gourmet Case Study: Requirements Phase 308
10.15 Air Gourmet Case Study: Rapid Prototype 311
10.16 Challenges of the Requirements Phase 313
Chapter Review 315
For Further Reading 315
Problems 316
References 317

Chapter 11
Specification Phase 319

11.1 The Specification Document 319
11.2 Informal Specifications 321
11.2.1 Case Study: Text Processing 322
11.3 Structured Systems Analysis 323
11.3.1 Sally's Software Shop 323
11.4 Other Semiformal Techniques 331
11.5 Entity-Relationship Modeling 332
11.6 Finite State Machines 335
11.6.1 Elevator Problem: Finite State Machines 336

11.7 Petri Nets 341
11.7.1 Elevator Problem: Petri Nets 343
11.8 Z 346
11.8.1 Elevator Problem: Z 347
11.8.2 Analysis of Z 349
11.9 Other Formal Techniques 351
11.10 Comparison of Specification Techniques 352
11.11 Testing during the Specification Phase 353
11.12 CASE Tools for the Specification Phase 354
11.13 Metrics for the Specification Phase 355
11.14 Air Gourmet Case Study: Structured Systems Analysis 355
11.15 Air Gourmet Case Study: Software Project Management Plan 357
11.16 Challenges of the Specification Phase 358
Chapter Review 358
For Further Reading 359
Problems 360
References 362

Chapter 12
Object-Oriented Analysis Phase 366

12.1 Object-Oriented Analysis 366
12.2 Elevator Problem: Object-Oriented Analysis 369
12.3 Use-Case Modeling 369
12.4 Class Modeling 371
12.4.1 Noun Extraction 372
12.4.2 CRC Cards 374
12.5 Dynamic Modeling 375
12.6 Testing during the Object-Oriented Analysis Phase 378
12.7 CASE Tools for the Object-Oriented Analysis Phase 383
12.8 Air Gourmet Case Study: Object-Oriented Analysis 383
12.9 Challenges of the Object-Oriented Analysis Phase 390
Chapter Review 391
For Further Reading 391
Problems 392
References 393

Chapter 13
Design Phase 395

13.1 Design and Abstraction 395
13.2 Action-Oriented Design 396
13.3 Data Flow Analysis 397
13.3.1 Data Flow Analysis Example 398
13.3.2 Extensions 402
13.4 Transaction Analysis 403
13.5 Data-Oriented Design 406
13.6 Object-Oriented Design 406
13.7 Elevator Problem: Object-Oriented Design 407
13.8 Formal Techniques for Detailed Design 415
13.9 Real-Time Design Techniques 416
13.10 Testing during the Design Phase 418
13.11 CASE Tools for the Design Phase 418
13.12 Metrics for the Design Phase 419
13.13 Air Gourmet Case Study: Object-Oriented Design 420
13.14 Challenges of the Design Phase 429
Chapter Review 429
For Further Reading 430
Problems 431
References 431

Chapter 14
Implementation Phase 434

14.1 Choice of Programming Language 434
14.2 Fourth-Generation Languages 437
14.3 Good Programming Practice 440
14.4 Coding Standards 445
14.5 Module Reuse 446

14.6 Module Test Case Selection 447
14.6.1 Testing to Specifications versus Testing to Code 447
14.6.2 Feasibility of Testing to Specifications 447
14.6.3 Feasibility of Testing to Code 448
14.7 Black-Box Module-Testing Techniques 451
14.7.1 Equivalence Testing and Boundary Value Analysis 451
14.7.2 Functional Testing 452
14.8 Glass-Box Module-Testing Techniques 454
14.8.1 Structural Testing: Statement, Branch, and Path Coverage 454
14.8.2 Complexity Metrics 456
14.9 Code Walkthroughs and Inspections 458
14.10 Comparison of Module-Testing Techniques 458
14.11 Cleanroom 459
14.12 Potential Problems When Testing Objects 460
14.13 Management Aspects of Module Testing 463
14.14 When to Rewrite Rather than Debug a Module 463
14.15 CASE Tools for the Implementation Phase 465
14.16 Air Gourmet Case Study: Black-Box Test Cases 465
14.17 Challenges of the Implementation Phase 467
Chapter Review 467
For Further Reading 468
Problems 469
References 470

Chapter 15
Implementation and Integration Phase 474

15.1 Introduction to Implementation and Integration 474
15.1.1 Top-down Implementation and Integration 475
15.1.2 Bottom-up Implementation and Integration 477
15.1.3 Sandwich Implementation and Integration 478
15.1.4 Implementation and Integration of Object-Oriented Products 480
15.1.5 Management Issues during the Implementation and Integration Phase 480
15.2 Testing during the Implementation and Integration Phase 481
15.3 Integration Testing of Graphical User Interfaces 481
15.4 Product Testing 482
15.5 Acceptance Testing 483
15.6 CASE Tools for the Implementation and Integration Phase 484
15.6 CASE Tools for the Complete Software Process 484
15.8 Integrated Environments 485
15.9 Environments for Business Applications 486
15.10 Public Tool Infrastructures 487
15.11 Potential Problems with Environments 487
15.12 Metrics for the Implementation and Integration Phase 488
15.13 Air Gourmet Case Study: Implementation and Integration Phase 488
15.14 Challenges of the Implementation and Integration Phase 489
Chapter Review 489
For Further Reading 490
Problems 490
References 492

Chapter 16
Maintenance Phase 493

16.1 Why Maintenance Is Necessary 493
16.2 What Is Required of Maintenance Programmers 494
16.3 Maintenance Case Study 497
16.4 Management of Maintenance 498
16.4.1 Fault Reports 498
16.4.2 Authorizing Changes to the Product 499

需要全本请在线购买: www.ertongbook.com