计 算 机 科 学 丛 书

Fourth Edition

# Software Engineering with JAVA

# 软件工程
## Java语言实现

（英文版·第4版）

(美) Stephen R. Schach 著
范德比尔特大学

机械工业出版社 China Machine Press
McGraw-Hill

计算机科学丛书

# 软件工程
# Java 语言实现

(英文版)

# Software Engineering with JAVA

(Fourth Edition)

(美) Stephen R. Schach 著
(范德比尔特大学)

机械工业出版社
China Machine Press

McGraw-Hill

Stephen R. Schach: Software Engineering with JAVA.

Copyright © 1998 by The McGraw-Hill Companies, Inc. All rights reserved. Jointly published by China Machine Press/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.

RISBN 007-1168729

本书英文影印版由 McGraw-Hill 公司授权机械工业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：01-99-0109**

**图书在版编目(CIP)数据**

软件工程：用Java语言描述：英文/(美)斯查(Schach, S. R.)著，－影印版．－北京：机械工业出版社，1999.3

(计算机科学丛书)

ISBN 7-111-06714-2

Ⅰ.软… Ⅱ.斯… Ⅲ.①软件工程－基本知识－英文 ②Java语言－程序设计－英文 Ⅳ.TP311.5

中国版本图书馆CIP数据核字 (1999) 第02379号

出 版 者：马九荣(北京市西城区百万庄大街22号 邮政编码 100037)

昌平环球印刷厂印刷．新华书店北京发行所发行

1999年3月第1版第1次印刷

787mm × 1092mm 1/16 · 39.75印张

印数：0001-5000册

定价：51.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

# PREFACE

Java is the premier language of the World Wide Web and hence of the Internet and the Information Super Highway. It is also the programming language du jour, the darling of the columnists in computer newspapers and e-zines (online magazines). Furthermore, as a consequence of the vast economic potential of the Information Super Highway, Java is probably the best known programming language among investors who may know nothing about computers; the name "Java" appears in *The Wall Street Journal* nearly as frequently as "pork-belly futures" and "selling short against the box."

These are most definitely *not* the reasons why I chose to bring out a Java version of *Classical and Object-Oriented Software Engineering*, Third Edition. On the contrary, the reason why I selected Java is that Java embodies the principles of software engineering. This is described in greater detail in the epilogue of this book, entitled "Java: A Case Study in Software Engineering."

Java is much more than just a language for World Wide Web applets (small application programs) that can be accessed over the Web and then run on one's own computer. On the contrary, Java is a general-purpose programming language that can be used for software for all kinds and, in addition, on the Web.

At the end of each chapter in the Third Edition of *Classical and Object-Oriented Software Engineering* there is a continuing major Case Study implemented in C++. In this book, that Case Study is implemented in Java, demonstrating that Java can be used for serious software. Certain of the problems in this book require the student to modify the Case Study in some way. The source code of the Case Study (Appendixes C and I) is available by anonymous ftp from ftp.vuse.vanderbilt.edu (address 129.59.100.10) in directory /pub /Software_Eng/Java, or on a diskette from Richard D. Irwin, 1333 Burr Ridge Parkway, Burr Ridge, IL 60521.

Also, students are required to use Java to implement the 14-part Term Project of this book. The detailed solution of the Term Project in the Instructor's Manual (available from Richard D. Irwin, at the address given at the end of the previous paragraph) is an additional demonstration that Java is indeed a general-purpose programming language.

## Acknowledgments

I am grateful to Rob Bland of IBM, co-author of the Instructor's Manual for this book, for his many helpful and insightful suggestions, especially regarding Chapter 6.

I should also like to thank the many individuals at Richard D. Irwin who have worked on this book. I am especially grateful to senior sponsoring editor Betsy Jones and senior project supervisor Becky Dodson.

Finally, I thank my family for their encouragement and unquestioning support throughout the writing of this book. After five other books, they fully appreciate that

I write best when I am alone in my study, even though I would far rather be with them. As always, I lovingly dedicate this book to my wife, Sharon, and my children, David and Lauren.

**Stephen R. Schach**

试读结束：需要全本请在线购买： www.ertongbook.com

# PREFACE TO THE THIRD EDITION OF *CLASSICAL AND OBJECT-ORIENTED SOFTWARE ENGINEERING*

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 CD 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


Prologue 1

PART 1
Introduction to the Software Process 3

CHAPTER 1
Scope of Software Engineering 5

CHAPTER 2
The Software Process and Its Problems 30

CHAPTER 3
Software Life-Cycle Models 53

CHAPTER 4
Stepwise Refinement, CASE, and Other Tools of the Trade 82

CHAPTER 5
Testing Principles 110

CHAPTER 6
Introduction to Objects 140

PART 2
The Phases of the Software Process 192

CHAPTER 7
Requirements Phase 197

CHAPTER 8
Specification Phase 222

CHAPTER 9
Object-Oriented Analysis Phase 268

CHAPTER 10
Planning Phase 291

CHAPTER 11
Design Phase 322

CHAPTER 12
Implementation Phase 368

CHAPTER 13
Implementation and Integration Phase 441

CHAPTER 14
Maintenance Phase 465

Epilogue 483

Appendices

APPENDIX A
Osbert Oglesby, Art Dealer 491

APPENDIX B
Software Engineering Resources 494

APPENDIX C
MSG Case Study: Rapid Prototype 496

APPENDIX D
MSG Case Study: Structured Systems Analysis 509




试读结束：需要全本请在线购买： www.ertongbook.com

APPENDIX E
MSG Case Study: Object-Oriented Analysis 513

APPENDIX F
MSG Case Study: Software Project Management Plan 514

APPENDIX G
MSG Case Study: Design 519

APPENDIX H
MSG Case Study: Black-Box Test Cases 539

APPENDIX I
MSG Case Study: Source Code 542

Bibliography 581

Author Index 605

Subject Index 608

# CONTENTS


**Prologue 1**

**PART 1**
**Introduction to the Software Process 3**

**CHAPTER 1**
**Scope of Software Engineering 5**

1.1 Historical Aspects 6
1.2 Economic Aspects 9
1.3 Maintenance Aspects 10
1.4 Specification and Design Aspects 14
1.5 Team Programming Aspects 16
1.6 The Object-Oriented Paradigm 17
1.7 Terminology 22
Chapter Review 24
For Further Reading 25
Problems 26
References 27

**CHAPTER 2**
**The Software Process and Its Problems 30**

2.1 Client, Developer, and User 32
2.2 Requirements Phase 33
2.2.1 Requirements Phase Testing 34
2.3 Specification Phase 35
2.3.1 Specification Phase Testing 36
2.4 Planning Phase 36
2.4.1 Planning Phase Testing 37
2.5 Design Phase 38
2.5.1 Design Phase Testing 39
2.6 Implementation Phase 39
2.6.1 Implementation Phase Testing 39
2.7 Integration Phase 40
2.7.1 Integration Phase Testing 40
2.8 Maintenance Phase 41
2.8.1 Maintenance Phase Testing 42
2.9 Retirement 42
2.10 Problems with Software Production: Essence and Accidents 43
2.10.1 Complexity 44
2.10.2 Conformity 46
2.10.3 Changeability 47
2.10.4 Invisibility 48
2.10.5 No Silver Bullet? 49
Chapter Review 50
For Further Reading 50
Problems 51
References 52

**CHAPTER 3**
**Software Life-Cycle Models 53**

3.1 Build-and-Fix Model 53
3.2 Waterfall Model 54
3.2.1 Analysis of the Waterfall Model 57
3.3 Rapid Prototyping Model 59
3.3.1 Integrating the Waterfall and Rapid Prototyping Models 61
3.4 Incremental Model 61
3.4.1 Analysis of the Incremental Model 63
3.5 Spiral Model 66
3.5.1 Analysis of the Spiral Model 70
3.6 Comparison of Life-Cycle Models 71
3.7 Capability Maturity Model 71
3.8 ISO 9000 75
Chapter Review 76
For Further Reading 77
Problems 78
References 78

**CHAPTER 4**
**Stepwise Refinement, CASE, and Other Tools of the Trade 82**

4.1 Stepwise Refinement 82
4.1.1 Stepwise Refinement Example 83

4.2 Cost–Benefit Analysis 89
4.3 CASE (Computer-Aided Software Engineering) 90
4.3.1 Taxonomy of CASE 90
4.4 Scope of CASE 92
4.5 Software Versions 96
4.5.1 Revisions 96
4.5.2 Variations 97
4.6 Configuration Control 98
4.6.1 Configuration Control during Product Maintenance 100
4.6.2 Baselines 101
4.6.3 Configuration Control during Product Development 101
4.7 Build Tools 102
4.8 Productivity Gains with CASE Technology 103
4.9 Software Metrics 103
Chapter Review 105
For Further Reading 105
Problems 106
References 108

CHAPTER 5
Testing Principles 110

5.1 Quality Issues 111
5.1.1 Software Quality Assurance 111
5.1.2 Managerial Independence 112
5.2 Nonexecution-Based Testing 113
5.2.1 Walkthroughs 113
5.2.2 Managing Walkthroughs 114
5.2.3 Inspections 115
5.2.4 Comparison of Inspections and Walkthroughs 117
5.2.5 Metrics for Inspections 118
5.3 Execution-Based Testing 118
5.4 What Should Be Tested? 119
5.4.1 Utility 120
5.4.2 Reliability 120
5.4.3 Robustness 121
5.4.4 Performance 121
5.4.5 Correctness 122
5.5 Testing versus Correctness Proofs 124
5.5.1 Example of a Correctness Proof 124
5.5.2 Correctness Proof Case Study 128
5.5.3 Correctness Proofs and Software Engineering 129
5.6 Who Should Perform Execution-Based Testing? 131
5.7 When Testing Stops 133
Chapter Review 134
For Further Reading 134
Problems 135
References 137

CHAPTER 6
Introduction to Objects 140

6.1 What Is a Module? 140
6.2 Cohesion 144
6.2.1 Coincidental Cohesion 145
6.2.2 Logical Cohesion 145
6.2.3 Temporal Cohesion 146
6.2.4 Procedural Cohesion 147
6.2.5 Communicational Cohesion 148
6.2.6 Informational Cohesion 148
6.2.7 Functional Cohesion 149
6.2.8 Cohesion Example 150
6.3 Coupling 151
6.3.1 Content Coupling 151
6.3.2 Common Coupling 151
6.3.3 Control Coupling 154
6.3.4 Stamp Coupling 154
6.3.5 Data Coupling 155
6.3.6 Coupling Example 156
6.4 Data Encapsulation 157
6.4.1 Data Encapsulation and Product Development 161
6.4.2 Data Encapsulation and Product Maintenance 163
6.5 Abstract Data Types 166
6.6 Information Hiding 168
6.7 Objects 171
6.8 Polymorphism and Dynamic Binding 175
6.9 Cohesion and Coupling of Objects 177
6.10 Reuse 178
6.10.1 Impediments to Reuse 179
6.11 Reuse Case Studies 180
6.11.1 Raytheon Missile Systems Division 180
6.11.2 Toshiba Software Factory 182
6.11.3 NASA Software 183

6.11.4 GTE Data Services 184
6.11.5 Hewlett-Packard 184
6.12 Reuse and Maintenance 185
6.13 Objects and Productivity 186
Chapter Review 188
For Further Reading 188
Problems 189
References 191

**PART 2**
**The Phases of the Software Process 195**

**CHAPTER 7**
**Requirements Phase 197**

7.1 Requirements Analysis Techniques 198
7.2 Rapid Prototyping 199
7.3 Human Factors 201
7.4 Rapid Prototyping as a Specification Technique 203
7.5 Reusing the Rapid Prototype 205
7.6 Other Uses of Rapid Prototyping 207
7.7 Management Implications of the Rapid Prototyping Model 208
7.8 Experiences with Rapid Prototyping 209
7.9 Joint Application Design 211
7.10 Comparison of Requirements Analysis Techniques 211
7.11 Testing during the Requirements Phase 212
7.12 CASE Tools for the Requirements Phase 212
7.13 Metrics for the Requirements Phase 213
7.14 MSG Case Study: Requirements Phase 214
7.15 MSG Case Study: Rapid Prototype 216
Chapter Review 217
For Further Reading 218
Problems 219
References 220

**CHAPTER 8**
**Specification Phase 222**

8.1 The Specification Document 222
8.2 Informal Specifications 224
8.2.1 Case Study: Text Processing 225
8.3 Structured Systems Analysis 226
8.3.1 Sally's Software Shop 226
8.4 Other Semiformal Techniques 234
8.5 Entity-Relationship Modeling 235
8.6 Finite State Machines 237
8.6.1 Elevator Problem: Finite State Machines 239
8.7 Petri Nets 244
8.7.1 Elevator Problem: Petri Nets 247
8.8 Z 250
8.8.1 Elevator Problem: Z 251
8.8.2 Analysis of Z 253
8.9 Other Formal Techniques 255
8.10 Comparison of Specification Techniques 256
8.11 Testing during the Specification Phase 256
8.12 CASE Tools for the Specification Phase 257
8.13 Metrics for the Specification Phase 258
8.14 MSG Case Study: Structured Systems Analysis 258
Chapter Review 260
For Further Reading 261
Problems 262
References 264

**CHAPTER 9**
**Object-Oriented Analysis Phase 268**

9.1 Object-Oriented versus Structured Paradigm 268
9.2 Object-Oriented Analysis 270
9.3 Elevator Problem: Object-Oriented Analysis 272
9.3.1 Class Modeling 272
9.3.2 Dynamic Modeling 275
9.3.3 Functional Modeling 278
9.4 Object-Oriented Life-Cycle Models 280
9.5 CASE Tools for the Object-Oriented Analysis Phase 282
9.6 MSG Case Study: Object-Oriented Analysis 283

Chapter Review 286
For Further Reading 286
Problems 288
References 289

**CHAPTER 10**
## Planning Phase 291

10.1 Estimating Duration and Cost 291
10.1.1 Metrics for the Size of a Product 293
10.1.2 Techniques of Cost Estimation 297
10.1.3 Intermediate COCOMO 299
10.1.4 Tracking Duration and Cost Estimates 303
10.2 Components of a Software Project Management Plan 303
10.3 Software Project Management Plan Framework 305
10.4 IEEE Software Project Management Plan 305
10.5 Planning of Testing 308
10.6 Planning of Object-Oriented Projects 310
10.7 Training Requirements 310
10.8 Documentation Standards 311
10.9 CASE Tools for the Planning Phase 312
10.10 Testing during the Planning Phase 315
10.11 MSG Case Study: Planning Phase 315
Chapter Review 315
For Further Reading 316
Problems 317
References 318

**CHAPTER 11**
## Design Phase 322

11.1 Design and Abstraction 322
11.2 Action-Oriented Design 324
11.3 Data Flow Analysis 324
11.3.1 Data Flow Analysis Example 325
11.3.2 Extensions 329
11.4 Transaction Analysis 329
11.5 Data-Oriented Design 332
11.6 Jackson System Development 333
11.6.1 Overview of Jackson System Development 333
11.6.2 Why Jackson System Development Is Presented in This Chapter 335
11.6.3 Elevator Problem: Jackson System Development 336
11.6.4 Analysis of Jackson System Development 344
11.7 Techniques of Jackson, Warnier, and Orr 345
11.8 Object-Oriented Design 346
11.8.1 Elevator Problem: Object-Oriented Design 347
11.9 Detailed Design 350
11.10 Comparison of Action-, Data-, and Object-Oriented Design 352
11.11 Difficulties Associated with Real-Time Systems 353
11.12 Real-Time Design Techniques 354
11.13 Testing during the Design Phase 355
11.14 CASE Tools for the Design Phase 356
11.15 Metrics for the Design Phase 357
11.16 MSG Case Study: Object-Oriented Design 358
Chapter Review 359
For Further Reading 361
Problems 363
References 364

**CHAPTER 12**
## Implementation Phase 368

12.1 Choice of Programming Language 368
12.2 Fourth Generation Languages 372
12.3 Structured Programming 375
12.3.1 History of Structured Programming 375
12.3.2 Why the goto Statement Is Considered Harmful 377
12.4 Good Programming Practice 378
12.5 Coding Standards 383
12.6 Team Organization 385
12.7 Democratic Team Approach 387
12.7.1 Analysis of the Democratic Team Approach 388
12.8 Classical Chief Programmer Team Approach 388
12.8.1 The *New York Times* Project 390

12.8.2 Impracticality of the Classical Chief Programmer Team Approach 391
12.9 Beyond Chief Programmer and Democratic Teams 392
12.10 Portability 396
12.10.1 Hardware Incompatibilities 396
12.10.2 Operating System Incompatibilities 398
12.10.3 Numerical Software Incompatibilities 398
12.10.4 Compiler Incompatibilities 399
12.11 Why Portability? 402
12.12 Techniques for Achieving Portability 404
12.12.1 Portable System Software 404
12.12.2 Portable Application Software 405
12.12.3 Portable Data 406
12.13 Module Reuse 407
12.14 Module Test Case Selection 407
12.14.1 Testing to Specifications versus Testing to Code 408
12.14.2 Feasibility of Testing to Specifications 408
12.14.3 Feasibility of Testing to Code 409
12.15 Black-Box Module-Testing Techniques 411
12.15.1 Equivalence Testing and Boundary Value Analysis 411
12.15.2 Functional Testing 413
12.16 Glass-Box Module-Testing Techniques 414
12.16.1 Structural Testing: Statement, Branch, and Path Coverage 414
12.16.2 Complexity Metrics 415
12.17 Code Walkthroughs and Inspections 418
12.18 Comparison of Module-Testing Techniques 418
12.19 Cleanroom 419
12.20 Testing Objects 420
12.21 Management Aspects of Module-Testing 423
12.21.1 When to Rewrite Rather Than Debug a Module 424
12.22 Testing Distributed Software 425
12.23 Testing Real-Time Software 427
12.24 CASE Tools for the Implementation Phase 429
12.25 MSG Case Study: Black-Box Test Cases 429
Chapter Review 431
For Further Reading 431
Problems 433
References 435

## CHAPTER 13
## Implementation and Integration Phase 441

13.1 Implementation and Integration 441
13.1.1 Top-Down Implementation and Integration 442
13.1.2 Bottom-Up Implementation and Integration 444
13.1.3 Sandwich Implementation and Integration 445
13.1.4 Implementation and Integration of Object-Oriented Products 446
13.1.5 Management Issues during the Implementation and Integration Phase 446
13.2 Testing during the Implementation and Integration Phase 447
13.3 Integration Testing of Graphical User Interfaces 447
13.4 Product Testing 448
13.5 Acceptance Testing 449
13.6 CASE Tools for the Implementation and Integration Phase 450
13.7 CASE Tools for the Complete Software Process 451
13.8 Language-Centered Environments 451
13.9 Structure-Oriented Environments 452
13.10 Toolkit Environments 452
13.11 Integrated Environments 452
13.11.1 Process Integration 453
13.11.2 Tool Integration 454
13.11.3 Other Forms of Integration 456
13.12 Environments for Business Applications 456
13.13 Public Tool Infrastructures 457
13.14 Comparison of Environment Types 458
13.15 Metrics for the Implementation and Integration Phase 458

试读结束：需要全本请在线购买： www.ertongbook.com