



STEPHEN R. SCHACH

Third Edition

C L A S S I C A L

a n d

O B J E C T - O R I E N T E D

S O F T W A R E

E N G I N E E R I N G

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*

©The McGraw-Hill Companies, Inc.,
1990, 1993, and 1996

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

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

The following
are registered
trademarks:

Access	Method/1
ADF	Microsoft
ADW	Motif
Aide-de-Camp	MS-DOS
Analyst/Designer	MVS/360
Apple	Natural
Bachman Product Set	Nomad
Battlemap	OMTool
Borland	1-800-FLOWERS
Bull	ORB Plus
CA-Tellaplan	OS/360
CCC	OS/370
Coca Cola	OS/VS2
CVS	Powerhouse
Demo II	QAPartner
Emeraude	RAMIS-II
Excel	Rational
Excelerator	Rose
Focus	SoftBench
Foundation	Software through Pictures
Ford	SPARCstation
FoxBASE	SQL
Guide	Statemate
Hewlett-Packard	Sun
Honeywell	System Architect
Hypercard	Teamwork
Hypertalk	The Design Machine
IBM	UNIX
IEW	VAX
IMS/360	VM/370
Informix	VMS
Lotus 1-2-3	Windows
Macintosh	X11
MacProject	XRunner

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

PART 1

**Introduction to the
Software Process 1**

CHAPTER 1
Scope of Software Engineering 3

CHAPTER 2
**The Software Process and
Its Problems 28**

CHAPTER 3
Software Life-Cycle Models 52

CHAPTER 4
**Stepwise Refinement, CASE, and
Other Tools of the Trade 81**

CHAPTER 5
Testing Principles 109

CHAPTER 6
Introduction to Objects 139

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

CHAPTER 7
Requirements Phase 197

CHAPTER 8
Specification Phase 222

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

CHAPTER 10
Planning Phase 290

CHAPTER 11
Design Phase 321

CHAPTER 12
Implementation Phase 367

CHAPTER 13
**Implementation and
Integration Phase 438**

CHAPTER 14
Maintenance Phase 462

Appendices

APPENDIX A
Osbert Oglesby, Art Dealer 481

APPENDIX B
**Software
Engineering Resources 484**

APPENDIX C
**MSG Case Study:
Rapid Prototype 486**

APPENDIX D
**MSG Case Study: Structured
Systems Analysis 496**

APPENDIX E
**MSG Case Study:
Object-Oriented Analysis 500**

APPENDIX F

**MSG Case Study: Software Project
Management Plan 501**

APPENDIX G

MSG Case Study: Design 506

APPENDIX H

**MSG Case Study: Black-Box
Test Cases 527**

APPENDIX I

**MSG Case Study: Source
Code 531**

Bibliography 564

Author Index 589

Subject Index 592

CONTENTS

PART 1

Introduction to the Software Process 1

CHAPTER 1

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 12
- 1.5 Team Programming Aspects 14
- 1.6 The Object-Oriented Paradigm 15
- 1.7 Terminology 20
- Chapter Review 22
- For Further Reading 22
- Problems 23
- References 25

CHAPTER 2

The Software Process and Its Problems 28

- 2.1 Client, Developer, and User 30
- 2.2 Requirements Phase 31
 - 2.2.1 Requirements Phase Testing 32
- 2.3 Specification Phase 33
 - 2.3.1 Specification Phase Testing 34
- 2.4 Planning Phase 34
 - 2.4.1 Planning Phase Testing 35
- 2.5 Design Phase 36
 - 2.5.1 Design Phase Testing 37
- 2.6 Implementation Phase 37
 - 2.6.1 Implementation Phase Testing 37
- 2.7 Integration Phase 38
 - 2.7.1 Integration Phase Testing 38
- 2.8 Maintenance Phase 40
 - 2.8.1 Maintenance Phase Testing 40
- 2.9 Retirement 41

2.10 Problems with Software Production:

- Essence and Accidents 41
- 2.10.1 Complexity 43
- 2.10.2 Conformity 44
- 2.10.3 Changeability 45
- 2.10.4 Invisibility 46
- 2.10.5 No Silver Bullet? 47

Chapter Review 48

For Further Reading 48

Problems 49

References 50

CHAPTER 3

Software Life-Cycle Models 52

- 3.1 Build-and-Fix Model 52
- 3.2 Waterfall Model 53
 - 3.2.1 Analysis of the Waterfall Model 56
- 3.3 Rapid Prototyping Model 58
 - 3.3.1 Integrating the Waterfall and Rapid Prototyping Models 60
- 3.4 Incremental Model 60
 - 3.4.1 Analysis of the Incremental Model 62
- 3.5 Spiral Model 65
 - 3.5.1 Analysis of the Spiral Model 69
- 3.6 Comparison of Life-Cycle Models 70
- 3.7 Capability Maturity Model 70
- 3.8 ISO 9000 74
- Chapter Review 75
- For Further Reading 76
- Problems 77
- References 77

CHAPTER 4

Stepwise Refinement, CASE, and Other Tools of the Trade 81

- 4.1 Stepwise Refinement 81
 - 4.1.1 Stepwise Refinement Example 82
- 4.2 Cost-Benefit Analysis 88

- 4.3 CASE (Computer-Aided Software Engineering) 89
 - 4.3.1 Taxonomy of CASE 89
- 4.4 Scope of CASE 91
- 4.5 Software Versions 95
 - 4.5.1 Revisions 95
 - 4.5.2 Variations 96
- 4.6 Configuration Control 97
 - 4.6.1 Configuration Control during Product Maintenance 99
 - 4.6.2 Baselines 100
 - 4.6.3 Configuration Control during Product Development 100
- 4.7 Build Tools 101
- 4.8 Productivity Gains with CASE Technology 102
- 4.9 Software Metrics 102
- Chapter Review 104
- For Further Reading 104
- Problems 105
- References 107

CHAPTER 5

Testing Principles 109

- 5.1 Quality Issues 110
 - 5.1.1 Software Quality Assurance 110
 - 5.1.2 Managerial Independence 111
- 5.2 Nonexecution-Based Testing 112
 - 5.2.1 Walkthroughs 112
 - 5.2.2 Managing Walkthroughs 113
 - 5.2.3 Inspections 114
 - 5.2.4 Comparison of Inspections and Walkthroughs 116
 - 5.2.5 Metrics for Inspections 117
- 5.3 Execution-Based Testing 117
- 5.4 What Should Be Tested? 118
 - 5.4.1 Utility 119
 - 5.4.2 Reliability 119
 - 5.4.3 Robustness 120
 - 5.4.4 Performance 120
 - 5.4.5 Correctness 121
- 5.5 Testing versus Correctness Proofs 123
 - 5.5.1 Example of a Correctness Proof 123
 - 5.5.2 Correctness Proof Case Study 127

- 5.5.3 Correctness Proofs and Software Engineering 128
- 5.6 Who Should Perform Execution-Based Testing? 130
- 5.7 When Testing Stops 132
- Chapter Review 133
- For Further Reading 133
- Problems 134
- References 136

CHAPTER 6

Introduction to Objects 139

- 6.1 What Is a Module? 139
- 6.2 Cohesion 143
 - 6.2.1 Coincidental Cohesion 144
 - 6.2.2 Logical Cohesion 144
 - 6.2.3 Temporal Cohesion 145
 - 6.2.4 Procedural Cohesion 146
 - 6.2.5 Communicational Cohesion 147
 - 6.2.6 Informational Cohesion 147
 - 6.2.7 Functional Cohesion 148
 - 6.2.8 Cohesion Example 148
- 6.3 Coupling 149
 - 6.3.1 Content Coupling 150
 - 6.3.2 Common Coupling 150
 - 6.3.3 Control Coupling 152
 - 6.3.4 Stamp Coupling 153
 - 6.3.5 Data Coupling 154
 - 6.3.6 Coupling Example 154
- 6.4 Data Encapsulation 157
 - 6.4.1 Data Encapsulation and Product Development 159
 - 6.4.2 Data Encapsulation and Product Maintenance 161
- 6.5 Abstract Data Types 166
- 6.6 Information Hiding 168
- 6.7 Objects 169
- 6.8 Polymorphism and Dynamic Binding 174
- 6.9 Cohesion and Coupling of Objects 176
- 6.10 Reuse 177
 - 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 181
 - 6.11.3 NASA Software 182

- 6.11.4 GTE Data Services 183
- 6.11.5 Hewlett-Packard 183
- 6.12 Reuse and Maintenance 184
- 6.13 Objects and Productivity 185
- Chapter Review 187
- For Further Reading 187
- Problems 189
- References 190

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 281
- 9.5 CASE Tools for the Object-Oriented Analysis Phase 283
- 9.6 MSG Case Study: Object-Oriented Analysis 283
- Chapter Review 287
- For Further Reading 287

Problems 287
References 289

CHAPTER 10

Planning Phase 290

- 10.1 Estimating Duration and Cost 290
 - 10.1.1 Metrics for the Size of a Product 292
 - 10.1.2 Techniques of Cost Estimation 296
 - 10.1.3 Intermediate COCOMO 298
 - 10.1.4 Tracking Duration and Cost Estimates 302
- 10.2 Components of a Software Project Management Plan 302
- 10.3 Software Project Management Plan Framework 304
- 10.4 IEEE Software Project Management Plan 304
- 10.5 Planning of Testing 307
- 10.6 Planning of Object-Oriented Projects 309
- 10.7 Training Requirements 309
- 10.8 Documentation Standards 310
- 10.9 CASE Tools for the Planning Phase 311
- 10.10 Testing during the Planning Phase 314
- 10.11 MSG Case Study: Planning Phase 314
- Chapter Review 314
- For Further Reading 315
- Problems 316
- References 317

CHAPTER 11

Design Phase 321

- 11.1 Design and Abstraction 321
- 11.2 Action-Oriented Design 323
- 11.3 Data Flow Analysis 323
 - 11.3.1 Data Flow Analysis Example 324
 - 11.3.2 Extensions 328
- 11.4 Transaction Analysis 328
- 11.5 Data-Oriented Design 331
- 11.6 Jackson System Development 332
 - 11.6.1 Overview of Jackson System Development 332
 - 11.6.2 Why Jackson System Development Is Presented in This Chapter 334

- 11.6.3 Elevator Problem: Jackson System Development 335
- 11.6.4 Analysis of Jackson System Development 343
- 11.7 Techniques of Jackson, Warnier, and Orr 344
- 11.8 Object-Oriented Design 345
 - 11.8.1 Elevator Problem: Object-Oriented Design 346
- 11.9 Detailed Design 349
- 11.10 Comparison of Action-, Data-, and Object-Oriented Design 351
- 11.11 Difficulties Associated with Real-Time Systems 352
- 11.12 Real-Time Design Techniques 353
- 11.13 Testing during the Design Phase 354
- 11.14 CASE Tools for the Design Phase 355
- 11.15 Metrics for the Design Phase 356
- 11.16 MSG Case Study: Object-Oriented Design 357
- Chapter Review 358
- For Further Reading 361
- Problems 363
- References 364

CHAPTER 12

Implementation Phase 367

- 12.1 Choice of Programming Language 367
- 12.2 Fourth Generation Languages 370
- 12.3 Structured Programming 373
 - 12.3.1 History of Structured Programming 373
 - 12.3.2 Why the goto Statement Is Considered Harmful 375
- 12.4 Good Programming Practice 377
- 12.5 Coding Standards 382
- 12.6 Team Organization 383
- 12.7 Democratic Team Approach 385
 - 12.7.1 Analysis of the Democratic Team Approach 386
- 12.8 Classical Chief Programmer Team Approach 387
 - 12.8.1 The *New York Times* Project 389
 - 12.8.2 Impracticality of the Classical Chief Programmer Team Approach 390

12.9	Beyond Chief Programmer and Democratic Teams	390
12.10	Portability	392
12.10.1	Hardware Incompatibilities	395
12.10.2	Operating System Incompatibilities	396
12.10.3	Numerical Software Incompatibilities	397
12.10.4	Compiler Incompatibilities	397
12.11	Why Portability?	399
12.12	Techniques for Achieving Portability	401
12.12.1	Portable System Software	402
12.12.2	Portable Application Software	402
12.12.3	Portable Data	404
12.13	Module Reuse	404
12.14	Module Test Case Selection	405
12.14.1	Testing to Specifications versus Testing to Code	405
12.14.2	Feasibility of Testing to Specifications	406
12.14.3	Feasibility of Testing to Code	406
12.15	Black-Box Module-Testing Techniques	408
12.15.1	Equivalence Testing and Boundary Value Analysis	409
12.15.2	Functional Testing	410
12.16	Glass-Box Module-Testing Techniques	411
12.16.1	Structural Testing: Statement, Branch, and Path Coverage	411
12.16.2	Complexity Metrics	413
12.17	Code Walkthroughs and Inspections	415
12.18	Comparison of Module-Testing Techniques	416
12.19	Cleanroom	416
12.20	Testing Objects	417
12.21	Management Aspects of Module Testing	420
12.21.1	When to Rewrite Rather than Debug a Module	421
12.22	Testing Distributed Software	422
12.23	Testing Real-Time Software	424
12.24	CASE Tools for the Implementation Phase	426
12.25	MSG Case Study: Black-Box Test Cases	427

Chapter Review	428
For Further Reading	429
Problems	430
References	432

CHAPTER 13

Implementation and Integration Phase 438

13.1	Implementation and Integration	438
13.1.1	Top-Down Implementation and Integration	439
13.1.2	Bottom-Up Implementation and Integration	441
13.1.3	Sandwich Implementation and Integration	442
13.1.4	Implementation and Integration of Object-Oriented Products	443
13.1.5	Management Issues during the Implementation and Integration Phase	443
13.2	Testing during the Implementation and Integration Phase	444
13.3	Integration Testing of Graphical User Interfaces	444
13.4	Product Testing	445
13.5	Acceptance Testing	446
13.6	CASE Tools for the Implementation and Integration Phase	447
13.7	CASE Tools for the Complete Software Process	447
13.8	Language-Centered Environments	448
13.9	Structure-Oriented Environments	448
13.10	Toolkit Environments	449
13.11	Integrated Environments	449
13.11.1	Process Integration	449
13.11.2	Tool Integration	450
13.11.3	Other Forms of Integration	453
13.12	Environments for Business Applications	453
13.13	Public Tool Infrastructures	454
13.14	Comparison of Environment Types	454
13.15	Metrics for the Implementation and Integration Phase	455
13.16	MSG Case Study: Implementation and Integration Phase	456

Chapter Review	457
For Further Reading	457
Problems	458
References	459

CHAPTER 14

Maintenance Phase 462

14.1	Why Maintenance Is Necessary	462
14.2	What Is Required of Maintenance Programmers	463
14.3	Maintenance Case Study	465
14.4	Management of Maintenance	466
14.4.1	Fault Reports	467
14.4.2	Authorizing Changes to the Product	468
14.4.3	Ensuring Maintainability	468
14.4.4	Problem of Repeated Maintenance	469
14.5	Maintenance of Object-Oriented Software	470
14.6	Maintenance Skills versus Development Skills	473
14.7	Reverse Engineering	473
14.8	Testing during the Maintenance Phase	474
14.9	CASE Tools for the Maintenance Phase	475
14.10	Metrics for the Maintenance Phase	476
	Chapter Review	476
	For Further Reading	477
	Problems	477
	References	478

Appendices

APPENDIX A	
Osbert Oglesby—Art Dealer	481

APPENDIX B	
Software	
Engineering Resources	484

APPENDIX C	
MSG Case Study:	
Rapid Prototype	486

APPENDIX D	
MSG Case Study: Structured	
Systems Analysis	496

APPENDIX E	
MSG Case Study:	
Object-Oriented Analysis	500

APPENDIX F	
MSG Case Study: Software Project	
Management Plan	501

APPENDIX G	
MSG Case Study: Design	506

APPENDIX H	
MSG Case Study: Black-Box	
Test Cases	527

APPENDIX I	
MSG Case Study: Source	
Code	531

Bibliography	564
---------------------	------------

Author Index	589
---------------------	------------

Subject Index	593
----------------------	------------