

SOFTWARE EVOLUTION AND MAINTENANCE

SOFTWARE EVOLUTION AND MAINTENANCE

A Practitioner's Approach

**PRIYADARSHI TRIPATHY
KSHIRASAGAR NAIK**

WILEY

Copyright © 2015 by John Wiley & Sons, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

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, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services please contact our Customer Care Department with the U.S. at 877-762-2974, outside the U.S. at 317-572-3993 or fax 317-572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print, however, may not be available in electronic format.

Library of Congress Cataloging-in-Publication Data

Tripathy, Priyadarshi, 1958–

Software evolution and maintenance : a practitioner's approach / Priyadarshi Tripathy,
Kshirasagar Naik.

pages cm

Includes index.

ISBN 978-0-470-60341-3 (cloth)

I. Software maintenance. I. Naik, Kshirasagar, 1959– II. Title.

QA76.76.S64T75 2015

005.1'6–dc23

2014033541

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

To our parents
Kunjabihari and Surekha Tripathy
Sukru and Teva Naik

CONTENTS

Preface	xiii
List of Figures	xvii
List of Tables	xxi
1 Basic Concepts and Preliminaries	1
1.1 Evolution Versus Maintenance, 1	
1.1.1 Software Evolution, 3	
1.1.2 Software Maintenance, 4	
1.2 Software Evolution Models and Processes, 6	
1.3 Reengineering, 9	
1.4 Legacy Systems, 11	
1.5 Impact Analysis, 12	
1.6 Refactoring, 13	
1.7 Program Comprehension, 14	
1.8 Software Reuse, 15	
1.9 Outline of the Book, 16	
References, 18	
Exercises, 23	
2 Taxonomy of Software Maintenance and Evolution	25
2.1 General Idea, 25	
2.1.1 Intention-Based Classification of Software Maintenance, 26	
2.1.2 Activity-Based Classification of Software Maintenance, 28	
2.1.3 Evidence-Based Classification of Software Maintenance, 28	

2.2	Categories of Maintenance Concepts,	37
2.2.1	Maintained Product,	37
2.2.2	Maintenance Types,	40
2.2.3	Maintenance Organization Processes,	41
2.2.4	Peopleware,	43
2.3	Evolution of Software Systems,	44
2.3.1	SPE Taxonomy,	46
2.3.2	Laws of Software Evolution,	49
2.3.3	Empirical Studies,	54
2.3.4	Practical Implications of the Laws,	56
2.3.5	Evolution of FOSS Systems,	58
2.4	Maintenance of Cots-Based Systems,	61
2.4.1	Why Maintenance of CBS Is Difficult?,	62
2.4.2	Maintenance Activities for CBSs,	65
2.4.3	Design Properties of Component-Based Systems,	67
2.5	Summary,	70
	Literature Review,	73
	References,	75
	Exercises,	80

3 Evolution and Maintenance Models 83

3.1	General Idea,	83
3.2	Reuse-Oriented Model,	84
3.3	The Staged Model for Closed Source Software,	87
3.4	The Staged Model for Free, Libre, Open Source Software,	90
3.5	Change Mini-Cycle Model,	91
3.6	IEEE/EIA Maintenance Process,	94
3.7	ISO/IEC 14764 Maintenance Process,	99
3.8	Software Configuration Management,	111
3.8.1	Brief History,	112
3.8.2	SCM Spectrum of Functionality,	113
3.8.3	SCM Process,	117
3.9	CR Workflow,	119
3.10	Summary,	125
	Literature Review,	126
	References,	129
	Exercises,	131

4 Reengineering 133

4.1	General Idea,	133
4.2	Reengineering Concepts,	135
4.3	A General Model for Software Reengineering,	137
4.3.1	Types of Changes,	140

- 4.3.2 Software Reengineering Strategies, 141
- 4.3.3 Reengineering Variations, 143
- 4.4 Reengineering Process, 144
 - 4.4.1 Reengineering Approaches, 144
 - 4.4.2 Source Code Reengineering Reference Model, 146
 - 4.4.3 Phase Reengineering Model, 150
- 4.5 Code Reverse Engineering, 153
- 4.6 Techniques Used for Reverse Engineering, 156
 - 4.6.1 Lexical Analysis, 157
 - 4.6.2 Syntactic Analysis, 157
 - 4.6.3 Control Flow Analysis, 157
 - 4.6.4 Data Flow Analysis, 158
 - 4.6.5 Program Slicing, 158
 - 4.6.6 Visualization, 160
 - 4.6.7 Program Metrics, 162
- 4.7 Decompilation Versus Reverse Engineering, 164
- 4.8 Data Reverse Engineering, 165
 - 4.8.1 Data Structure Extraction, 168
 - 4.8.2 Data Structure Conceptualization, 169
- 4.9 Reverse Engineering Tools, 170
- 4.10 Summary, 174
- Literature Review, 176
- References, 178
- Exercises, 185

5 Legacy Information Systems

187

- 5.1 General Idea, 187
- 5.2 Wrapping, 189
 - 5.2.1 Types of Wrapping, 189
 - 5.2.2 Levels of Encapsulation, 191
 - 5.2.3 Constructing a Wrapper, 192
 - 5.2.4 Adapting a Program for Wrapper, 194
 - 5.2.5 Screen Scraping, 194
- 5.3 Migration, 195
- 5.4 Migration Planning, 196
- 5.5 Migration Methods, 202
 - 5.5.1 Cold Turkey, 202
 - 5.5.2 Database First, 203
 - 5.5.3 Database Last, 204
 - 5.5.4 Composite Database, 205
 - 5.5.5 Chicken Little, 206
 - 5.5.6 Butterfly, 208
 - 5.5.7 Iterative, 212
- 5.6 Summary, 217

Literature Review, 218
References, 219
Exercises, 221

6 Impact Analysis 223

- 6.1 General Idea, 223
 - 6.2 Impact Analysis Process, 225
 - 6.2.1 Identifying the SIS, 228
 - 6.2.2 Analysis of Traceability Graph, 229
 - 6.2.3 Identifying the Candidate Impact Set, 231
 - 6.3 Dependency-Based Impact Analysis, 234
 - 6.3.1 Call Graph, 234
 - 6.3.2 Program Dependency Graph, 235
 - 6.4 Ripple Effect, 238
 - 6.4.1 Computing Ripple Effect, 238
 - 6.5 Change Propagation Model, 242
 - 6.5.1 Recall and Precision of Change Propagation Heuristics, 243
 - 6.5.2 Heuristics for Change Propagation, 245
 - 6.5.3 Empirical Studies, 246
 - 6.6 Summary, 247
- Literature Review, 248
References, 249
Exercises, 253

7 Refactoring 255

- 7.1 General Idea, 255
- 7.2 Activities in a Refactoring Process, 258
 - 7.2.1 Identify What to Refactor, 258
 - 7.2.2 Determine Which Refactorings Should be Applied, 259
 - 7.2.3 Ensure that Refactoring Preserves the Behavior of the Software, 261
 - 7.2.4 Apply the Refactorings to the Chosen Entities, 262
 - 7.2.5 Evaluate the Impacts of the Refactorings on Quality, 263
 - 7.2.6 Maintain Consistency of Software Artifacts, 265
- 7.3 Formalisms for Refactoring, 265
 - 7.3.1 Assertions, 265
 - 7.3.2 Graph Transformation, 266
 - 7.3.3 Software Metrics, 267
- 7.4 More Examples of Refactorings, 271
- 7.5 Initial Work on Software Restructuring, 273
 - 7.5.1 Factors Influencing Software Structure, 273
 - 7.5.2 Classification of Restructuring Approaches, 275
 - 7.5.3 Restructuring Techniques, 276
- 7.6 Summary, 282

Literature Review, 283
 References, 286
 Exercises, 288

8 Program Comprehension **289**

8.1 General Idea, 289
 8.2 Basic Terms, 291
 8.2.1 Goal of Code Cognition, 291
 8.2.2 Knowledge, 291
 8.2.3 Mental Model, 293
 8.2.4 Understanding Code, 296
 8.3 Cognition Models for Program Understanding, 298
 8.3.1 Letovsky Model, 298
 8.3.2 Shneiderman and Mayer Model, 301
 8.3.3 Brooks Model, 303
 8.3.4 Soloway, Adelson, and Ehrlich Model, 308
 8.3.5 Pennington Model, 310
 8.3.6 Integrated Metamodel, 312
 8.4 Protocol Analysis, 315
 8.5 Visualization for Comprehension, 317
 8.6 Summary, 321
 Literature Review, 321
 References, 322
 Exercises, 324

9 Reuse and Domain Engineering **325**

9.1 General Idea, 325
 9.1.1 Benefits of Reuse, 327
 9.1.2 Reuse Models, 327
 9.1.3 Factors Influencing Reuse, 328
 9.1.4 Success Factors of Reuse, 329
 9.2 Domain Engineering, 329
 9.2.1 Draco, 331
 9.2.2 DARE, 331
 9.2.3 FAST, 331
 9.2.4 FORM, 331
 9.2.5 Kobra, 332
 9.2.6 PLUS, 332
 9.2.7 PuLSE, 332
 9.2.8 Koala, 332
 9.2.9 RSEB, 332
 9.3 Reuse Capability, 333
 9.4 Maturity Models, 334
 9.4.1 Reuse Maturity Model, 334

9.4.2	Reuse Capability Model, 336	
9.4.3	RiSE Maturity Model, 338	
9.5	Economic Models of Software Reuse, 340	
9.5.1	Cost Model of Gaffney and Durek, 346	
9.5.2	Application System Cost Model of Gaffney and Cruickshank, 348	
9.5.3	Business Model of Poulin and Caruso, 350	
9.6	Summary, 352	
	Literature Review, 352	
	References, 353	
	Exercises, 356	
Glossary		359
Index		379

PREFACE

karmany eva dhikaras te; ma phalesu kadachana; ma karmaphalahetur bhur; ma te sango
stv akarmani.

Your right is to work only; but never to the fruits thereof; may you not be motivated by
the fruits of actions; nor let your attachment to be towards inaction.

—Bhagavad Gita

We have been witnessing stellar growth of the global software industry for three decades. As this century progresses the industry is engaged in fixing defects and enhancing and adding new features to the existing software applications. In fact, more resources are spent on software maintenance than on actual software development. The imbalance between software development and maintenance is opening up new business opportunities for software off-shoring companies. It is also generating much research interest to develop methods and tools for improving software evolution and maintenance (SEAM).

Twenty-five years ago, the software industry was a much smaller one, and the academia used to offer a single, comprehensive course entitled *Software Engineering* to educate undergraduate students in the nuts and bolts of software development and maintenance. Although software maintenance has been a part of the classical software engineering literature for decades, the subject has not been widely incorporated into the mainstream undergraduate curriculum. A few universities have started offering an *option* in software engineering comprising four specialized courses, namely, *Requirements Specification*, *Software Design*, *Software Testing and Quality Assurance*, and *Software Evolution and Maintenance*. In addition, some universities have introduced full undergraduate and graduate degree programs in software engineering.

Our survey of the subject of software evolution and maintenance reveals that a large body of work exists in disparate form, including research papers, technical reports, and reports of working groups. Moreover, there are many excellent books focusing on specific aspects of a course in software maintenance and evolution. However, there is no single book that presents the materials in a comprehensive manner. Absence of a comprehensive textbook explaining most of the aspects of software evolution and maintenance creates several problems for instructors and students alike. For example, an instructor needs to refer to many sources to prepare lecture materials. Consequently, it takes much time on the part of the instructor, and students do not have access to all those sources. Our goal is to introduce the students and the instructors to a set of well-rounded educational materials covering the fundamental developments in software evolution and common maintenance practices in the industry. We intend to provide students a single, comprehensive textbook covering most of the topics in evolution and maintenance with much detail so that it is very easy to get a handle on SEAM without reading a number of books and articles in this subject. We have not tried to specifically address their research challenges. Instead, we have presented the evolution theory and practice as a broad stepping stone which will enable the students and practitioners to understand and develop maintenance practices for complex software system.

We decided to write this book based on our teaching, research, and industrial experiences in software maintenance. For the past 20 years, Sagar has been teaching software engineering, including software testing and maintenance on a regular basis; Piyu managed software quality assurance teams in industry for the maintenance of routers, switches, wireless data networks, storage networks, and intrusion prevention appliances. Our combined experience has helped us in selecting and structuring the contents of this book to make it suitable as a text.

WHO SHOULD READ THIS BOOK?

We have written this book to introduce students, researchers, and software professionals to the fundamental developments in evolution models and common maintenance practices for software. Undergraduate students in software engineering, computer science, and computer engineering will be introduced to the subject matter in a step-by-step manner. Practitioners too will benefit from the structured presentation and comprehensive nature of the materials. Graduate students can use it as a reference. After reading the whole book, the reader will have gained a thorough understanding of the following topics:

- Laws of software evolution and the means to control them
- Evolution and maintenance models, including maintenance of commercial off-the-shelf systems
- Reengineering techniques and processes for migration of legacy information systems

- Impact analysis and change propagation techniques
- Program comprehension and refactoring
- Reuse and domain engineering models

Each chapter gives a clear understanding of a particular topic in software evolution by discussing the main ideas with examples. It starts by explaining the basic concepts about the topic, thereby ensuring a common base of understanding; next, it expands the presentation by drilling the important aspects deeper.

HOW SHOULD THIS BOOK BE READ?

This book consists of several independent topics in SEAM glued together. Chapters 1, 2, and 3 provide basic understanding of the subject matters. Therefore, the first three chapters must be read in order. Next, depending upon the interest of the reader, one can choose any chapter to study without any difficulty. However, we recommend the reader to study Chapters 4 and 5 together in that order. This is because Chapter 4 (“Reengineering”) introduces basic concepts of reengineering, reverse engineering, and data reverse engineering, whereas Chapter 5 (“Legacy Information Systems”) discusses the migration of a system after it is reengineered. Therefore, in our opinion, the ordering will facilitate easier understanding of the materials, especially for those who are new to software evolution.

Notes for instructors

The book can be used as a text in an introductory course in SEAM. It is desirable to cover all the chapters in an introductory course in SEAM. When used as a recommended text in a software engineering course, the following selected portions can help students imbibe the essential concepts in software evolution and maintenance:

- Chapter 1: All the sections
- Chapter 2: Sections 2.1 and 2.3
- Chapter 3: Sections 3.1, 3.2, 3.3, 3.4, 3.5 and 3.8
- Chapter 4: Sections 4.1 and 4.2
- Chapter 5: Sections 5.1 and 5.2
- Chapter 6: All the sections
- Chapter 7: Sections 7.1 and 7.2
- Chapter 8: Sections 8.1 and 8.2
- Chapter 9: Section 9.1

Supplementary materials for instructors are available at: <http://ece.uwaterloo.ca/~snaik/mybook2.html>

ACKNOWLEDGMENTS

While preparing this book, we received invaluable support of different kinds from many people, including researchers, the publisher, our family members, our friends, and our colleagues. First, we thank all the researchers who have been shaping this field ever since programs were started to be written. Without their published work, this book would not have seen the light of the day. Second, we thank our editors, namely, George Telecki, Michael Christian, and Whitney A. Lesch, who gave us much professional guidance and patiently answered our various questions. The first author, Piyu Tripathy, would like to thank his former colleagues at Cisco Systems, Airvana Inc., NEC Laboratories America Inc., and present colleagues at Knowledge Trust.

Finally, the supports of our parents, parents-in-law, and spouses deserve a special mention. I, Piyu Tripathy, thank my dear wife Leena, who has taken many household and family duties off my hands to give me time that I needed to write this book; I would like to thank my newly arrived daughter Inu for asking me inquisitive questions about this book, which helped me in writing this preface.

I, Sagar Naik, thank my loving wife Alaka for her invaluable support. I also thank my charming daughters, Monisha and Sameeksha, and exciting son, Siddharth, for their understanding while I was writing this book. Finally, I heartily acknowledge all the support that my elder brother Gajapati extended to me. We are very pleased that now we have more time for our families.

PRIYADARSHI (PIYU) TRIPATHY

*Knowledge Trust
Bhubaneswar, India*

KSHIRASAGAR (SAGAR) NAIK

*University of Waterloo
Waterloo, Canada*

LIST OF FIGURES

2.1	Groups or clusters and their types	29
2.2	Decision tree types. From Reference 15. © 2001 John Wiley & Sons	32
2.3	Overview of concept categories affecting software maintenance	38
2.4	Inputs and outputs of software evolution. From Reference 26. © 1988 John Wiley & Sons	45
2.5	S-type programs	47
2.6	P-type programs	48
2.7	E-type programs	49
2.8	E-type programs with feedback. From Reference 33. © 2006 John Wiley & Sons	50
2.9	Onion model of FOSS development structure	59
2.10	Growth of the major subsystems (development releases only) of the Linux OS. From Reference 57. © 2000 IEEE	61
3.1	Traditional SDLC model. From Reference 1. © 1988 John Wiley & Sons	84
3.2	The quick fix model. From Reference 2. © 1990 IEEE	85
3.3	The iterative enhancement model. From Reference 2. © 1990 IEEE	85
3.4	The full reuse model. From Reference 2. © 1990 IEEE	86
3.5	The simple staged model for the CSS life cycle. From Reference 6. © 2000 IEEE	88
3.6	The versioned staged model for the CSS life cycle. From Reference 6. © 2000 IEEE	90
3.7	The staged model for the FLOSS system. From Reference 9. © 2007 ACM	91

3.8	The change min-cycle. From Reference 12. © 2008 Springer	92
3.9	Seven phases of IEEE maintenance process. From Reference 26. © 2004 IEEE	95
3.10	Problem identification phase	96
3.11	Analysis phase	97
3.12	Design phase	97
3.13	Implementation phase	98
3.14	System test phase	98
3.15	Acceptance test phase	99
3.16	Delivery phase	100
3.17	ISO/IEC 14764 iterative maintenance process. From Reference 26. © 2004 IEEE	102
3.18	Process implementation activity	102
3.19	Problem and modification activity	103
3.20	Modification implementation activity	105
3.21	Maintenance review/acceptance activity	106
3.22	Migration activity	107
3.23	Retirement activity	109
3.24	Technical dimensions of SCM systems	113
3.25	An evolution of a file with two branches	114
3.26	A process for implementing SCM	117
3.27	State transition diagram of a CR	120
4.1	Levels of abstraction and refinement. From Reference 5. © 1992 IEEE	136
4.2	Conceptual basis for the reengineering process. From Reference 5. © 1992 IEEE	137
4.3	General model of software reengineering. From Reference 5. © 1992 IEEE	138
4.4	Horseshoe model of reengineering. From Reference 7. © 1998 IEEE	139
4.5	Conceptual basis for reengineering strategies. From Reference 5. © 1992 IEEE	142
4.6	Source code reengineering reference model. From Reference 15. © 1990 IEEE	147
4.7	The interface nomenclature. From Reference 15. © 1990 IEEE. “(N)-” represents the N th layer	147
4.8	Software reengineering process phases. From Reference 14. © 1992 IEEE	150
4.9	Replacement strategies for recoding	152
4.10	Relationship between reengineering and reverse engineering. From Reference 6. © 1990 IEEE	154
4.11	A block of code to compute the sum and product of all the even integers in the range $[0, N)$ for $N \geq 3$	159
4.12	The backward slice of code obtained from Figure 4.11 by using the criterion $S < [7]$; sum >	159

4.13	The forward slice of code obtained from Figure 4.11 by using the criterion $S < [3]$; product $>$	160
4.14	Relationship between decompilation and traditional reengineering. From Reference 83. © 2007	165
4.15	General architecture of the DBRE methodology. From Reference 95. © 1997 IEEE	169
4.16	Basic structure of reverse engineering tools. From Reference 6. © 1990 IEEE	171
5.1	Forward wrapper. From Reference 10. © 2006 ACM	190
5.2	Backward wrapper. From Reference 10. © 2006 ACM	190
5.3	Levels of encapsulation. From Reference 11. © 1996 IEEE	191
5.4	Modules of a wrapping framework	193
5.5	Portfolio analysis chi-square chart	198
5.6	Database first approach. From Reference 19. © 1999 IEEE	203
5.7	Database last approach. From Reference 19. © 1999 IEEE	204
5.8	Composite database approach. From Reference 19. © 1999 IEEE	205
5.9	Application gateway. From Reference 19. © 1999 IEEE	206
5.10	Information system gateway. From Reference 19. © 1999 IEEE	207
5.11	Migrating TempStore in Butterfly methodology. From Reference 19. © 1999 IEEE	211
5.12	The iterative system architecture methodology during reengineering. From Reference 29. © 2003 IEEE	214
5.13	The iterative migration process. From Reference 29. © 2003 IEEE	215
6.1	Impact analysis process. From Reference 6. © 2008 IEEE	226
6.2	Traceability in software work products. From Reference 22. © 1991 IEEE	229
6.3	Underlying graph for maintenance. From Reference 22. © 1991 IEEE	230
6.4	Determine work product impact. From Reference 22. © 1991 IEEE	231
6.5	Simple directed graph of SLOs. From Reference 12. © 2002 IEEE	232
6.6	In-degree and out-degree of SLO1. From Reference 12. © 2002 IEEE	232
6.7	Example of a call graph. From Reference 26. © 2003 IEEE	234
6.8	Execution trace	235
6.9	Example program. From Reference 31. © 1990 ACM	236
6.10	Program dependency graph of the program in Figure 6.9	236
6.11	Dynamic program slice for the code in Figure 6.9, text case $X = -1$, with respect to variable Y	237
6.12	Intramodule and intermodule change propagation. From Reference 36. © 2001 John Wiley & Sons	239
6.13	Change propagation model. From Reference 10. © 2004 IEEE	243
6.14	Change propagation flow for a simple example. From Reference 10. © 2004 IEEE	244

6.15	Program	253
7.1	Class diagram of a local area network (LAN) simulator. From Reference 6. © 2007 Springer	260
7.2	Applications of two refactorings. From Reference 6. © 2007 Springer	263
7.3	An example of a soft-goal graph for maintainability, with one leaf node. From Reference 11. © 2002 IEEE	264
7.4	An example of a program graph. From Reference 13. © 2006 Elsevier	266
7.5	Program graph obtained after applying <i>push-down method</i> refactoring to the program graph of Figure 7.4. From Reference 13. © 2006 Elsevier	267
7.6	An example of a VRML diagram of two classes C1 and C2. Circles denote methods and squares denote attributes	269
7.7	Illustration of the push-down method refactoring: (a) the class diagram before refactoring; (b) the class diagram after refactoring	272
7.8	An example of parameterizing a method. There are four methods in (a), whereas there is one method in (b) with one parameter	273
7.9	Factors which can influence software structure. From Reference 2. © 1989 IEEE	274
7.10	Broad classification of approaches to software structuring	275
7.11	System sandwich approach to software restructuring. The arrows represent the flow of data and/or commands	278
7.12	Illustration of system level remodularization. Bullets represent low level entities. Dotted shapes represent modules. Arrows represent progression from one level to the next	279
7.13	Illustration of entity level remodularization. Bullets represent low level entities. Dotted shapes represent modules	279
7.14	Dendogram representation of Figure 7.12	282
8.1	Gaining general knowledge and software-specific knowledge	292
8.2	Letovsky's program comprehension model	298
8.3	Shneiderman and Mayer program comprehension model	302
8.4	An overview of Brooks comprehension model	304
8.5	Soloway, Adelson, and Ehrlich comprehension model	309
8.6	Pennington model	310
8.7	Integrated Metamodel. From Reference 1. © 1995 IEEE	313
9.1	Feedback between domain and application engineering	330
9.2	Reuse capability. From Reference 40. © 1993 IEEE	334

LIST OF TABLES

2.1	Evidence-Based 12 Mutually Exclusive Maintenance Types	30
2.2	Impact of the Types	31
2.3	Summary of Evidence-Based Types of Software Maintenance	33
2.4	Staged Model Maintenance Task	39
2.5	Laws of Software Evolution	50
2.6	System Data to be Used in Question 14	81
3.1	Template of a Maintenance Plan	101
3.2	Modification Request Task Steps	104
3.3	Option Task Steps	104
3.4	Documentation Task Steps	105
3.5	Review and Approval Task Steps	106
3.6	Migration Plan Task Steps	108
3.7	Operation and Training Task Steps	108
3.8	Retirement Plan Task Steps	110
3.9	Change Request Schema Field Summary	121
3.10	Engineering Change Document Information	124
4.1	Reengineering Process Variations	143
4.2	Tasks—Analysis and Planning Phase	151
4.3	Commonly Used Software Metrics	163
5.1	Common Quantifiable Benefit Metrics	200
5.2	Chicken Little Migration Approach	207
5.3	Phases of Butterfly Methodology	209
5.4	Migration Activities in Phase 1	209
5.5	Migration Activities in Phase 2	210
5.6	Migration Activities in Phase 3	210

5.7	Migration Activities in Phase 4	210
5.8	Migration Activities in Phase 5	211
6.1	Relationships Represented by a Connectivity Matrix	233
6.2	Relationships Represented by a Reachability Matrix	233
6.3	Relationship with Distance Indicators	234
6.4	Laws of Software Evolution	242
6.5	Performance of Change Propagation Heuristics for the Five Software Systems	247
8.1	Tasks and Activities Requiring Code Understanding	290
8.2	Code Cognition Models	290
9.1	Reuse Maturity Model	335
9.2	Critical Success Factors	337
9.3	RiSE Maturity Model Levels: Organizational Factors [42]	341
9.4	RiSE Maturity Model Levels: Business Factors [42]	343
9.5	RiSE Maturity Model Levels: Technological Factors [42]	344
9.6	RiSE Maturity Model Levels: Processes Factors [42]	345
9.7	Relative Costs of Development Activities	347
9.8	Relative Reuse Cost (b)	348