

Second Edition

Software Engineering

Modern Approaches



Eric J. Braude • Michael E. Bernstein

Software Engineering

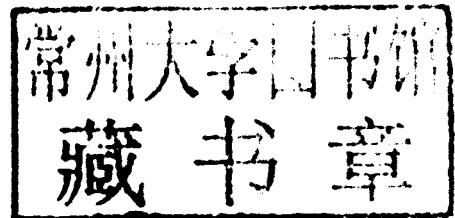
Modern Approaches
SECOND EDITION

Eric J. Braude

Boston University, Metropolitan College

Michael E. Bernstein

Boston University, Metropolitan College



JOHN WILEY & SONS, INC.

Executive Editor	Beth Lang Golub
Editor	Jonathan Shipley
Assistant Editor	Georgia King
Editorial Assistant	Mike Berlin
Marketing Manager	Christopher Ruel
Designer	RDC Publishing Group Sdn Bhd
Production Manager	Janis Soo
Assistant Production Editor	Yee Lyn Song

Cover Credit: © Hulton Archive/Getty Images

This book was set in 10/12 Point Weiss by Thomson Digital, and printed and bound by R.R. Donnelley. The cover was printed by R.R. Donnelley.

This book is printed on acid free paper. ∞

Copyright © 2011, 2001 John Wiley & Sons, Inc. 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, scanning or otherwise, except as permitted under Sections 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, website 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 07030-5774, (201)748-6011, fax (201)748-6008, website <http://www.wiley.com/go/permissions>.

Evaluation copies are provided to qualified academics and professionals for review purposes only, for use in their courses during the next academic year. These copies are licensed and may not be sold or transferred to a third party. Upon completion of the review period, please return the evaluation copy to Wiley. Return instructions and a free of charge return shipping label are available at www.wiley.com/go/returnlabel. Outside of the United States, please contact your local representative.

Library of Congress Cataloging-in-Publication Data

Braude, Eric J.

Software engineering : modern approaches / Eric J. Braude, Michael E. Bernstein. — 2nd ed.

p. cm.

Includes bibliographical references and index.

ISBN 978-0-471-69208-9 (cloth)

1. Software engineering. 2. Object-oriented programming (Computer science) I. Bernstein, Michael E. II. Title. QA76.758.B74 2011 005.1—dc22

2009051247

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

For Judy (Eric J. Braude)
To Bambi, Garrett and Reid,
for all their love and support (Michael E. Bernstein)

Preface

Much of the modern world runs on software. As a result, software engineers are entrusted with significant responsibility. Although it is a biomedical engineer, for example, who designs health monitoring systems, it is a software engineer who creates its actual control functions. A marketing professional develops ways to reach customers online but it is a software engineer who makes the system a reality.

Today's software engineer must be able to participate in more than one kind of software process, work in agile teams, deal with customers, express requirements clearly, create modular designs, utilize legacy and open source projects, monitor quality, incorporate security, and apply many types of tests.

THE ISSUE OF SCALE

A software application consists of tens, hundreds, even thousands of classes. This is very different from managing three or four of them, and results in the dragon of complexity suggested by this book's cover. As also suggested there, however, this dragon can be subdued. Indeed, to deal with numerous and complex classes, software engineers have at their disposal a wide variety of tools and techniques. These range from the waterfall process to agile methodologies, from highly integrated tool suites to refactoring and loosely coupled tool sets. Underlying this variety is continuing research into reliable approaches, and an acknowledgment of the fact that one size does not fit all projects.

THIS EDITION COMPARED WITH THE FIRST

The first edition of this book emphasized the object-oriented approach, which has subsequently become widespread. It was also designed to help student teams carry out hands-on term projects through theory, examples, case studies, and practical steps. Object-orientation and hands-on continue to be major features of this edition. However, we have widened the scope of the first edition, especially by including extensive coverage of agile methods and refactoring, together with deeper coverage of quality and software design.

Readers of the first edition made extensive use of the complete video game case study—an example that they could follow “from soup to nuts” but which was significantly more comprehensive than a toy. This edition retains and updates that case study, but it adds the exploration of a simpler example on one hand (a DVD rental store) and large, real, open source case studies on the other. In particular, to provide students a feeling for the scope and complexity of real-world applications, this book leads them through selected requirements, design, implementation, and maintenance of the Eclipse and OpenOffice open source projects. The size, complexity, and transparency of these projects provide students a window into software engineering on a realistic scale.

Every book on software engineering faces a dilemma: how to reconcile the organization of the *topics* with the organization of actual software project *phases*. An organization of chapters into process/project management/requirements analysis/design/implementation/test/maintenance is straightforward but is liable to be misinterpreted as promoting the waterfall development process at the expense others. Our approach has been to use this organization in the seven parts of the book but to demonstrate throughout that each phase

typically belongs to a cycle rather than to a single waterfall sequence. In particular, our approach integrates agile methodologies consistently.

This edition also introduces somewhat advanced influential ideas, including model-driven architectures and aspect-oriented programming. Nowadays, formal methods are mandated by government agencies for the highest levels of security, and this book aims to educate readers in their possibilities. Due to print space limitations, some of this material is to be found in the online extension of this book.

In summary, specific features of this edition compared with the first are as follows:

- A sharpening and standardization of the material from the first edition
- A strong agile thread throughout, including a chapter on agility alone and one devoted to refactoring.
- A separate chapter on quality in six of the book's seven parts
- Real-world case studies, taken from the Eclipse and OpenOffice open source projects
- Greatly expanded coverage of software design and design patterns
- New chapters on advanced, influential software engineering ideas
- An organization of many of the book's seven parts as follows:
 - Principles
 - Details
 - Quality
 - Advanced Methods

HOW INSTRUCTORS CAN USE THIS BOOK

This book has been designed to accommodate multiple approaches to the learning and teaching of software engineering. Most instructors teach the fundamentals of software process, project management, requirements analysis, design, testing, implementation, and maintenance. Beyond this common ground, however, instructors employ a wide variety of styles and emphases. The following are major approaches, together with the sequence of chapters that support each of them.

- A. *Process emphasis*, concentrating on how applications are developed
All of Parts I through IV; and Chapters 15, 22, and 25 (the remaining principles and introduction chapters)
- B. *Design emphasis*, which teaches software engineering primarily as a design activity
Principles and introduction: Chapters 1, 3, 7, and 10; all of Part V; and Chapters 22 and 25 (principles and introduction)
- C. *Programming and agile emphasis*, which emphasizes software engineering as a code-oriented activity that satisfies requirements, emphasizing agile approaches
Principles and introduction: Chapters 1, 3, 7, 10, and 15; all of Part VI; and Chapters 25 and 26
- D. *Two-semester course*, which enables the instructor to cover most topics and assign a substantial hands-on project

- D1. All of the chapters in the book, either in sequence from beginning to end
or
- D2. In two passes as follows:
- (i) Principles and introduction chapters in the first semester: Chapters 1, 3, 7, 15, 22, and 25
 - (ii) The remaining chapters in the second semester
- E. *Emphasis on a hands-on projects and case studies*, which relies mostly on an active team or individual project as the vehicle for learning theory and principles
Principles and introduction chapters: Chapters 1, 3, 7, 15, 22, 25, and 26, and all case study sections in the remaining chapters
- F. *Theory and principles emphasis*, concentrating on what one can learn about software engineering and its underpinnings
Principles and introduction chapters: Chapters 1, 2, 3, 7, 15, 22, and 25, followed, as time allows, by Chapters 14 and 21 (emerging topics)
- G. *Quality assurance and testing emphasis*
Principles and introduction: Chapters 1, 3, 7, and 10; Chapters 2, 5, 9, 13, 20, 23 (quality); and Chapters 25, 26, 27, and 28 (testing).

The web site for this book, including review questions and the Encounter game case study, is www.wiley.com/college/braude.

Eric Braude
Michael Bernstein
Boston, MA
January 2010

Acknowledgments

We owe a debt of gratitude to our students at Boston University's Metropolitan College. Working in myriad industries and businesses, they have given us invaluable feedback. The College itself has provided a model place for the teaching and learning software engineering. Our thanks go to Dick Bostwick and Tom VanCourt, much of whose work in the first edition carries over to this one. We are grateful to the people of Wiley for working with us through the painstaking process of writing and publishing this book. We are particularly appreciative of the help from our editors, Dan Sayre and Jonathan Shipley; from Georgia King, Yee Lyn Song, and the indefatigable staff. We thank the reviewers of our manuscript, whose feedback has been invaluable:

Arvin Agah, University of Kansas
Steven C. Shaffer, Pennsylvania State University
Stephen M. Thebaut, University of Florida
Aravinda P. Sistla, University of Illinois, Chicago
James P. Purtilo, University of Maryland
Linda M. Ott, Michigan Technological University
Jianwei Niu, University of Texas, San Antonio
William Lively, Texas A&M University
Chung Lee, California State University, Pomona
Sudipto Ghosh, Colorado State University
Max I. Fomitchev, Pennsylvania State University
Lawrence Bernstein, Stevens Institute of Technology
John Dalbey, California Polytechnic University
Len Fisk, California State University, Chico
Ahmed M. Salem, California State University, Sacramento
Fred Strauss, New York University
Kai H. Chang, Auburn University
Andre van der Hoek, University of California, Irvine
Saeed Monemi, California Polytechnic University
Robert M. Cubert, University of Florida
Chris Tseng, San Jose State University
Michael James Payne, Purdue University
Carol A. Wellington, Shippensburg University
Yifei Dong, University of Oklahoma
Peter Blanchfield, Nottingham University
Desmond Greer, Queen's University Belfast
WeiQi Yan, Queen's University Belfast
Zaigham Mahmood, Derby University
Karel Pieterse, Hogeschool Van Amsterdam

This book would not have been possible without the constant love, patience, and encouragement of our families.

Brief Contents

Preface xiv

Acknowledgments xvii

Part I Introduction to Software Engineering

- 1 The Goals and Terminology of Software Engineering 1
- 2 Introduction to Quality and Metrics in Software Engineering 21

Part II Software Process

- 3 Software Process 32
- 4 Agile Software Processes 63
- 5 Quality in the Software Process 80
- 6 Software Configuration Management 120

Part III Project Management

- 7 Principles of Software Project Management I 140
- 8 Principles of Software Project Management II 168
- 9 Quality and Metrics in Project Management 213

Part IV Requirement Analysis

- 10 Principles of Requirements Analysis 230
- 11 Analyzing High-Level Requirements 245
- 12 Analyzing Detailed Requirements 278
- 13 Quality and Metrics in Requirements Analysis 331
- 14 Formal and Emerging Methods in Requirements Analysis (Online chapter) 349

Part V Software Design

- 15 Principles of Software Design 350
- 16 The Unified Modeling Language 361
- 17 Software Design Patterns 383
- 18 Software Architecture 438
- 19 Detailed Design 476
- 20 Design Quality and Metrics 508
- 21 Advanced and Emerging Methods in Software Design (Online chapter) 538

Part VI Implementation

- 22 Principles of Implementation 539
- 23 Quality and Metrics in Implementation 584
- 24 Refactoring 601

Part VII Testing and Maintenance

- 25 Introduction to Software Testing 621
- 26 Unit Testing 630
- 27 Module and Integration Testing 666
- 28 Testing at the System Level 694
- 29 Software Maintenance 730

Glossary 759

Index 767

Contents

Preface	xiv
The Issue of Scale	xiv
This Edition Compared with the First	xiv
How Instructors Can Use This Book	xv
Acknowledgments	xvii
 PART I Introduction to Software Engineering	
1 The Goals and Terminology of Software Engineering	1
1.1 What is Software Engineering	2
1.2 Why Software Engineering Is Critical: Software Disasters	3
1.3 Why Software Fails or Succeeds	4
1.4 Software Engineering Activities	5
1.5 Software Engineering Principles	10
1.6 Ethics in Software Engineering	12
1.7 Case Studies	14
1.8 Summary	19
1.9 Exercises	19
Bibliography	20
2 Introduction to Quality and Metrics in Software Engineering	21
2.1 The Meaning of Software Quality	22
2.2 Defects in Software	23
2.3 Verification and Validation	25
2.4 Planning for Quality	27
2.5 Metrics	28
2.6 Summary	30
2.7 Exercises	31
Bibliography	31
 PART II Software Process	
3 Software Process	32
3.1 The Activities of Software Process	33
3.2 Software Process Models	37
3.3 Case Study: Student Team Guidance	55

3.4	Summary	59
3.5	Exercises	60
	Bibliography	62
4	Agile Software Processes	63
4.1	Agile History and Agile Manifesto	64
4.2	Agile Principles	65
4.3	Agile Methods	66
4.4	Agile Processes	68
4.5	Integrating Agile with Non-Agile Processes	74
4.6	Summary	77
4.7	Exercises	78
	Bibliography	79
5	Quality in the Software Process	80
5.1	Principles of Managing Quality	81
5.2	Managing Quality in Agile Processes	82
5.3	Quality Planning	83
5.4	Inspections	87
5.5	QA Reviews and Audits	92
5.6	Defect Management	93
5.7	Process Improvement and Process Metrics	96
5.8	Organization-Level Quality and the CMMI	100
5.9	Case Study: Software Quality Assurance Plan for Encounter	103
5.10	Summary	118
5.11	Exercises	118
	Bibliography	119
6	Software Configuration Management	120
6.1	Software Configuration Management Goals	121
6.2	SCM Activities	121
6.3	Configuration Management Plans	128
6.4	Configuration Management Systems	128
6.5	Case Study: Encounter Video Game	129
6.6	Case Study: Eclipse	134
6.7	Student Team Guidance: Configuration Management	136
6.8	Summary	137
6.9	Exercises	138
	Bibliography	139
PART III Project Management		
7	Principles of Software Project Management I: Organization, Tools, and Risk Management	140
7.1	Software Project Organization	142
7.2	Team Size	144
7.3	Geographically Distributed Development	146
7.4	The Team Software Process	151
7.5	Software Project Tools and Techniques	153

7.6	Risk Management	159
7.7	Student Team Guidance: Organizing the Software Project's Management	162
7.8	Summary	165
7.9	Exercises	166
	Bibliography	167
8	Principles of Software Project Management II: Estimation, Scheduling, and Planning	168
8.1	Cost Estimation	169
8.2	Scheduling	182
8.3	The Software Project Management Plan	185
8.4	Case Study: Encounter Project Management Plan	187
8.5	Case Study: Project Management in Eclipse	196
8.6	Case Study: Project Management for OpenOffice	205
8.7	Case Study: Student Team Guidance	208
8.8	Summary	210
8.9	Exercises	211
	Bibliography	212
9	Quality and Metrics in Project Management	213
9.1	Cultivating and Planning Internal Quality	214
9.2	Project Metrics	215
9.3	Using Metrics for Improvement	219
9.4	Software Verification and Validation Plan	223
9.5	Case Study: Software Verification and Validation Plan for Encounter	225
9.6	Summary	228
9.7	Exercises	228
	Bibliography	229
PART IV Requirement Analysis		
10	Principles of Requirements Analysis	230
10.1	The Value of Requirements Analysis	231
10.2	Sources of Requirements	231
10.3	<i>High-level vs. Detailed</i> Requirements	232
10.4	Types of Requirements	233
10.5	Nonfunctional Requirements	233
10.6	Documenting Requirements	238
10.7	Traceability	239
10.8	Agile Methods and Requirements	239
10.9	Updating the Project to Reflect Requirements Analysis	241
10.10	Summary	243
10.11	Exercises	244
	Bibliography	244
11	Analyzing High-Level Requirements	245
11.1	Examples of Customer Wants	246
11.2	Stakeholder Vision	247
11.3	The Interview and Documentation Process	248

11.4	Writing an Overview	249
11.5	Describing Main Functions and Use Cases	249
11.6	Agile Methods for High-Level Requirements	252
11.7	Specifying User Interfaces: High Level	254
11.8	Security Requirements	258
11.9	Using Diagrams for High-Level Requirement	260
11.10	Case Study: High-Level Software Requirements Specification (SRS) for the Encounter Video Game	264
11.11	Case Study: High-Level Requirements for Eclipse	268
11.12	Eclipse Platform Subproject (First of three)	269
11.13	Case Study: High-Level Requirements for OpenOffice	273
11.14	Summary	275
11.15	Exercises	275
	Bibliography	276
12	Analyzing Detailed Requirements	278
12.1	The Meaning of Detailed Requirements	279
12.2	Organizing Detailed Requirements	280
12.3	User Interfaces: Detailed Requirements	291
12.4	Detailed Security Requirements	296
12.5	Error Conditions	296
12.6	Traceability of Detailed Requirements	297
12.7	Using Detailed Requirements to Manage Projects	300
12.8	Prioritizing Requirements	301
12.9	Associating Requirements with Tests	302
12.10	Agile Methods for Detailed Requirements	303
12.11	Using Tools and the Web for Requirements Analysis	305
12.12	The Effects on Projects of the Detailed Requirements Process	308
12.13	Student Project Guide: Requirements for the Encounter Case Study	309
12.14	Case Study: Detailed Requirements for the Encounter Video Game	315
12.15	Summary	328
12.16	Exercises	329
	Bibliography	330
13	Quality and Metrics in Requirements Analysis	331
13.1	Quality of Requirements for Agile Projects	332
13.2	Accessibility of Requirements	332
13.3	Comprehensiveness of Requirements	333
13.4	Understandability of Requirements	335
13.5	Unambiguity of Requirements	335
13.6	Consistency of Requirements	336
13.7	Prioritization of Requirements	337
13.8	Security and High-Level Requirements	338
13.9	Self-Completeness of Requirements	339
13.10	Testability of Requirements	340
13.11	Traceability of Requirements	342
13.12	Metrics for Requirements Analysis	343

13.13 Inspecting Detailed Requirements	344
13.14 Summary	347
13.15 Exercises	348
14 Formal and Emerging Methods in Requirements Analysis: An Introduction	
(Online Chapter)	349
14.1 Provable Requirements Method	
14.2 Introduction to Formal Methods	
14.3 Mathematical Preliminaries	
14.4 The Z-Specification Language	
14.5 The B Language System	
14.6 Trade-offs for Using a B-like system	
14.7 Summary	
14.8 Exercises	
Bibliography	
PART V Software Design	
15 Principles of Software Design	350
15.1 The Goals of Software Design	351
15.2 Integrating Design Models	354
15.3 Frameworks	357
15.4 IEEE Standards for Expressing Designs	359
15.5 Summary	359
15.6 Exercises	360
16 The Unified Modeling Language	361
16.1 Classes in UML	362
16.2 Class Relationships in UML	362
16.3 Multiplicity	364
16.4 Inheritance	364
16.5 Sequence Diagrams	368
16.6 State Diagrams	372
16.7 Activity Diagrams	374
16.8 Data Flow Models	376
16.9 A Design Example with UML	377
16.10 Summary	380
16.11 Exercises	381
Bibliography	382
17 Software Design Patterns	383
17.1 Examples of a Recurring Design Purpose	384
17.2 An Introduction to Design Patterns	386
17.3 Summary of Design Patterns by Type: Creational, Structural, and Behavioral	390
17.4 Characteristics of Design Patterns: Viewpoints, Roles, and Levels	396
17.5 Selected Creational Design Patterns	400
17.6 Selected Structural Design Patterns	408

17.7	Selected Behavioral Design Patterns	417
17.8	Design Pattern Forms: Delegation and Recursion	431
17.9	Summary	435
17.10	Exercises	436
	Bibliography	437
18	Software Architecture	438
18.1	A Categorization of Architectures	439
18.2	Software Architecture Alternatives and Their Class Models	439
18.3	Trading Off Architecture Alternatives	453
18.4	Tools for Architectures	454
18.5	IEEE Standards for Expressing Designs	455
18.6	Effects of Architecture Selection on the Project Plan	455
18.7	Case Study: Preparing to Design Encounter (Student Project Guide continued)	457
18.8	Case Study: Software Design Document for the <i>Role-Playing Video Game</i> Framework	460
18.9	Case Study: Software Design Document for Encounter (Uses the Framework)	462
18.10	Case Study: Architecture of Eclipse	466
18.11	Case Study: OpenOffice Architecture	468
18.12	Summary	473
18.13	Exercises	474
	Bibliography	475
19	Detailed Design	476
19.1	Relating Use Cases, Architecture, and Detailed Design	477
19.2	A Typical Road Map for the "Detailed Design" Process	478
19.3	Object-Oriented Design Principles	479
19.4	Designing against Interfaces	481
19.5	Specifying Classes, Functions, and Algorithms	482
19.6	Reusing Components	485
19.7	Sequence and Data Flow Diagrams for Detailed Design	486
19.8	Detailed Design and Agile Processes	490
19.9	Design in the Unified Development Process	490
19.10	IEEE Standard 890 for Detailed Design	491
19.11	Updating a Project with Detailed Design	491
19.12	Case Study: Detailed Design of Encounter	494
19.13	Case Study: Detailed Design of Eclipse	503
19.14	Summary	505
19.15	Exercises	505
	Bibliography	507
20	Design Quality and Metrics	508
20.1	Degree of Understandability, Cohesion, and Coupling	510
20.2	Degree of Sufficiency as a Quality Goal	510
20.3	Degree of Robustness as a Quality Goal	511
20.4	Degree of Flexibility as a Design Quality Goal	512
20.5	Degree of Reusability as a Design Quality Goal	513
20.6	Degree of Time Efficiency as a Design Quality Measure	517

20.7	Degree of Space Efficiency as a Design Quality Measure	519
20.8	Degree of Reliability as a Design Quality Measure	521
20.9	Degree of Security as a Design Quality Measure	523
20.10	Assessing Quality in Architecture Selection	525
20.11	Assessing the Quality of Detailed Designs	531
20.12	Summary	536
20.13	Exercises	536
	Bibliography	537
21	Advanced and Emerging Methods in Software Design (Online Chapter)	538
21.1	Designing in a Distributed Environment	
21.2	Introduction to Aspect-Oriented Programming	
21.3	Designing for Security with UMLsec	
21.4	Model-Driven Architectures	
21.5	The Formal Design Process in <i>B</i>	
21.6	Summary	
21.7	Exercises	
	Bibliography	
PART VI	Implementation	
22	Principles of Implementation	539
22.1	Agile and Non-Agile Approaches to Implementation	540
22.2	Choosing a Programming Language	540
22.3	Identifying Classes	540
22.4	Defining Methods	541
22.5	Implementation Practices	544
22.6	Defensive Programming	548
22.7	Coding Standards	552
22.8	Comments	554
22.9	Tools and Environments for Programming	555
22.10	Case Study: Encounter Implementation	556
22.11	Case Study: Eclipse	559
22.12	Case Study: OpenOffice	559
22.13	Student Team Guidance for Implementation	565
22.14	Summary	566
22.15	Code Listings Referred to in this Chapter	566
22.16	Exercises	581
	Bibliography	583
23	Quality and Metrics in Implementation	584
23.1	Quality of Implementation	585
23.2	Code Inspections and Related Quality Procedures	597
23.3	Summary	599
23.4	Exercises	599

24 Refactoring	601
24.1 Big Refactorings	604
24.2 Composing Methods	606
24.3 Moving Features between Objects	608
24.4 Organizing Data	609
24.5 Generalization	612
24.6 Introducing Modules	616
24.7 Refactoring in Projects	617
24.8 Summary	619
24.9 Exercises	619
Bibliography	620
 PART VII Testing and Maintenance	
25 Introduction to Software Testing	621
25.1 Testing Early and Often, and the Agile Connection	622
25.2 Retesting: Regression Testing	622
25.3 Black Box and White Box Testing	623
25.4 Unit Testing vs. Post-Unit Testing	624
25.5 Testing Object-Oriented Implementations	625
25.6 Documenting Tests	626
25.7 Test Planning	626
25.8 Testing Test Suites by Fault Injection	628
25.9 Summary	628
25.10 Exercises	629
 26 Unit Testing	630
26.1 The Sources of Units for Unit Testing	631
26.2 Unit Test Methods	631
26.3 Testing Methods	642
26.4 Test-Driven Development	647
26.5 Case Study: Encounter Video Game	652
26.6 Summary	662
26.7 Exercises	663
Bibliography	665
 27 Module and Integration Testing	666
27.1 Stubs and Drivers	667
27.2 Testing a Class	668
27.3 Integration	672
27.4 Daily Builds	679
27.5 Interface Testing	680
27.6 Module Integration	682
27.7 Case Study: Class Test for Encounter	683
27.8 Case Study: Encounter Integration Plan	688
27.9 Summary	692
27.10 Exercises	692
Bibliography	693