

Applying UML and Patterns

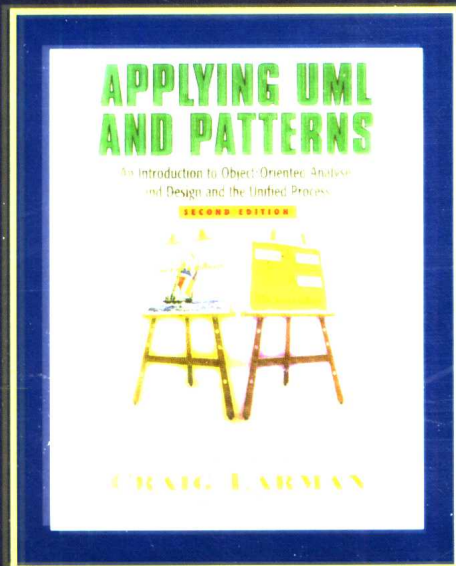
An Introduction to Object-Oriented Analysis and Design and the Unified Process

UML 和模式应用

面向对象分析和设计及统一过程导论

(第二版·影印版)

[美] Craig Larman 著
Philippe Kruchten 序



世界上最畅销介绍面向对象分析/设计、迭代开发和UML的书籍!

“经常有人问我,介绍面向对象设计的最好的书是什么。迄今为止,《UML和模式应用》这本书是我最好的选择。”

—— Martin Fowler,

《UML Distilled》与《重构》作者

...book to introduce them to the world of OO design.
...UML and Patterns has been my unreserved choice.
...Martin Fowler, author, UML Distilled and Refactoring

LARMAN

by Philippe Kruchten



中国电力出版社

www.infopower.com.cn

原 版 风 暴 系 列

Applying UML and Patterns

An Introduction to Object-Oriented Analysis and Design
and the Unified Process

UML 和模式应用

面向对象分析和设计及统一过程导论

(第二版·影印版)



中国电力出版社
www.infopower.com.cn

Applying UML and Patterns: an introduction to object-oriented analysis and design and the Unified Process,
Second Edition (ISBN 0-13-092569-1)

Craig Larman

Copyright © 2002 Prentice Hall PTR

Original English Language Edition Published by Prentice Hall PTR

All rights reserved.

Reprinting edition published by PEARSON EDUCATION NORTH ASIA LTD and CHINA ELECTRIC
POWER PRESS, Copyright © 2004.

本书影印版由 Pearson Education 授权中国电力出版社在中国境内（香港、澳门特别行政区和台湾地区除外）独家出版、发行。

未经出版者书面许可，不得以任何方式复制或抄袭本书的任何部分。

本书封面贴有 Pearson Education 防伪标签，无标签者不得销售。

北京市版权局著作合同登记号 图字：01-2004-1302

For sale and distribution in the People's Republic of China exclusively (except Taiwan, Hong Kong SAR and Macao SAR).

仅限于中华人民共和国境内（不包括中国香港、澳门特别行政区和中国台湾地区）销售发行。

图书在版编目（CIP）数据

UML 和模式应用：面向对象分析和设计及统一过程导论：第 2 版 / （美）拉尔曼（Larman, C.）著，
——影印本. 北京：中国电力出版社，2004

（原版风暴系列）

ISBN 7-5083-2204-5

I. U... II. 拉... III. 面向对象语言, UML—程序设计—英文 IV. TP312

中国版本图书馆 CIP 数据核字（2004）第 029862 号

丛 书 名：原版风暴系列

书 名：UML 和模式应用：面向对象分析和设计及统一过程导论（第二版·影印版）

编 著：（美）Craig Larman

责任编辑：牛贵华

出版发行：中国电力出版社

地址：北京市三里河路6号 邮政编码：100044

电话：（010）88515918 传 真：（010）88518169

印 刷：北京丰源印刷厂

开 本：787×1092 1/16 印 张：40.75 插页：2

书 号：ISBN 7-5083-2204-5

版 次：2004 年 5 月北京第 1 版 2004 年 5 月第 1 次印刷

定 价：59.80 元

版权所有 翻印必究

For Julie

Without your support, this would not have been possible.

For Haley and Hannah

Thanks for putting up with a distracted Daddy, again!

“This edition contains Larman’s usual accurate and thoughtful writing. It is a very good book made even better.”

—**Alistair Cockburn**, author, *Writing Effective Use Cases* and *Surviving OO Projects*.

“People often ask me which is the best book to introduce them to the world of OO design. Ever since I came across it *Applying UML and Patterns* has been my unre-served choice.”

—**Martin Fowler**, author, *UML Distilled* and *Refactoring*.

“This book makes learning UML enjoyable and pragmatic by incrementally introducing it as an intuitive language for specifying the artifacts of object analysis and design. It is a well written introduction to UML and object methods by an expert practitioner.”

—**Cris Kobryn**, key contributor to UML 1.x specifications, and chair of the UML Revision Task Force and UML 2.0 Working Group.

“Too few people have a knack for explaining things. Fewer still have a handle on software analysis and design. Craig Larman has both.”

—**John Vlissides**, author, *Design Patterns* and *Pattern Hatching*.

FOREWORD

Programming is fun, but developing quality software is hard. In between the nice ideas, the requirements or the “vision,” and a working software product, there is much more than programming. Analysis and design, defining how to solve the problem, what to program, capturing this design in ways that are easy to communicate, to review, to implement, and to evolve is what lies at the core of this book. This is what you will learn.

The Unified Modeling Language (UML) has become the universally-accepted language for software design blueprints. UML is the visual language used to convey design ideas throughout this book, which emphasizes how developers really apply frequently used UML elements, rather than obscure features of the language.

The importance of patterns in crafting complex systems has long been recognized in other disciplines. Software design patterns are what allow us to describe design fragments, and reuse design ideas, helping developers leverage the expertise of others. Patterns give a name and form to abstract heuristics, rules and best practices of object-oriented techniques. No reasonable engineer wants to start from a blank slate, and this book offers a palette of readily usable design patterns.

But software design looks a bit dry and mysterious when not presented in the context of a software engineering process. And on this topic, I am delighted that for his second edition, Craig Larman has chosen to embrace and introduce the Unified Process, showing how it can be applied in a relatively simple and low-ceremony way. By presenting the case study in an iterative, risk-driven, architecture-centric process, Craig’s advice has realistic context; he exposes the dynamics of what really happens in software development, and shows the external forces at play. The design activities are connected to other tasks, and they no longer appear as a purely cerebral activity of systematic transformations or creative intuition. And Craig and I are convinced of the benefits of iterative development, which you will see abundantly illustrated throughout.

So for me, this book has the right mix of ingredients. You will learn a systematic method to do Object-Oriented Analysis and Design (OOA/D) from a great teacher, a brilliant methodologist, and an “OO guru” who has taught it to thousands around the world. Craig describes the method in the context of the Uni-

FOREWORD

fied Process. He gradually presents more sophisticated design patterns—this will make the book very handy when you are faced with real-world design challenges. And he uses the most widely accepted notation.

I'm honored to have had the opportunity to work directly with the author of this major book. I enjoyed reading the first edition, and was delighted when he asked me to review the draft of his second edition. We met several times and exchanged many e-mails. I have learned much from Craig, even about our own process work on the Unified Process and how to improve it and position it in various organizational contexts. I am certain that you will learn a lot, too, in reading this book, even if you are already familiar with OOAD. And, like me, you will find yourself going back to it, to refresh your memory, or to gain further insights from Craig's explanations and experience.

In an iterative process, the result of the second iteration improves on the first. Similarly, the writing matures, I suppose; even if you have the first edition, you'll enjoy and benefit from the second one.

Happy reading!

Philippe Kruchten
Rational Fellow
Rational Software Canada
Vancouver, BC

PREFACE

Thank you for reading this book! This is a practical introduction to object-oriented analysis and design (OOA/D), and to related aspects of iterative development. I am grateful that the first edition was received as a popular introduction to OOA/D throughout the world, translated into many languages. Therefore, this second edition builds upon and refines—rather than replaces—the content in the first. I want to sincerely thank all the readers of the first edition.

Here is how the book will benefit you.

Design robust and maintainable object systems.

First, the use of object technology has proliferated in the development of software, and mastery of OOA/D is critical for you to create robust and maintainable object systems.

Follow a roadmap through requirements, analysis, design, and coding.

Second, if you are new to OOA/D, you are understandably challenged about how to proceed through this complex subject; this book presents a well-defined roadmap—the Unified Process—so that you can move in a step-by-step process from requirements to code.

Use the UML to illustrate analysis and design models.

Third, the Unified Modeling Language (UML) has emerged as the standard notation for modeling; so it is useful for you to be conversant in it. This book teaches the skills of OOA/D using the UML notation.

Improve designs by applying the “gang-of-four” and GRASP design patterns.

Fourth, design patterns communicate the “best practice” idioms and solutions that object-oriented design experts apply in order to create systems. In this book you will learn to apply design patterns, including the popular “gang-of-four” patterns, and the GRASP patterns which communicate fundamental principles of responsibility assignment in object design. Learning and applying patterns will accelerate your mastery of analysis and design.

Learn efficiently by following a refined presentation.

Fifth, the structure and emphasis in this book is based on years of experience in training and mentoring thousands of people in the art of OOA/D. It reflects that experience by providing a refined, proven, and efficient approach to learning the subject so your investment in reading and learning is optimized.

Learn from a realistic exercise.

Sixth, it exhaustively examines a single case study—to realistically illustrate the entire OOA/D process, and goes deeply into thorny details of the problem; it is a realistic exercise.

Translate to code.

Seventh, it shows how to map object design artifacts to code in Java.

Design a layered architecture.

Eighth, it explains how to design a layered architecture and relate the graphical user interface layer to domain and technical services layers.

Design a framework.

Finally, it shows you how to design an object-oriented framework and applies this to the creation of a framework for persistent storage in a database.

Objectives

The overarching objective is this:

Help students and developers create object designs through the application of a set of explainable principles and heuristics.

By studying and applying the information and techniques presented here, you will become more adept at understanding a problem in terms of its processes and concepts, and designing a solid solution using objects.

Intended Audience

This book is an *introduction* to OOA/D, related requirements analysis, and to iterative development with the Unified Process as a sample process; it is not meant as an advanced text. It is for the following audience:

- Developers and students with experience in an object-oriented programming language, but who are new—or relatively new—to object-oriented analysis and design.
- Students in computer science or software engineering courses studying object technology.
- Those with some familiarity in OOA/D who want to learn the UML notation, apply patterns, or who want to sharpen and deepen their analysis and design skills.

Prerequisites

Some prerequisite knowledge is assumed—and necessary—to benefit from this book:

- Knowledge and experience in an object-oriented programming language such as Java, C#, C++, or Smalltalk.
- Knowledge of fundamental object technology concepts, such as class, instance, interface, polymorphism, encapsulation, interfaces, and inheritance.

Fundamental object technology concepts are not defined.

Java Examples

In general, the book presents code examples in Java or discusses Java implementations, due to its widespread familiarity. However, the ideas presented are applicable to most—if not all—object-oriented programming languages.

Book Organization

The overall strategy in the organization of this book is that analysis and design topics are introduced in an order similar to that of a software development project running across an “inception” phase (a Unified Process term) followed by three iterations (see Figure P.1).

1. The inception phase chapters introduce the basics of requirements analysis.
2. Iteration 1 introduces fundamental OOA/D and how to assign responsibilities to objects.
3. Iteration 2 focuses on object design, especially on introducing some high-use “design patterns.”
4. Iteration 3 introduces a variety of subjects, such as architectural analysis and framework design.

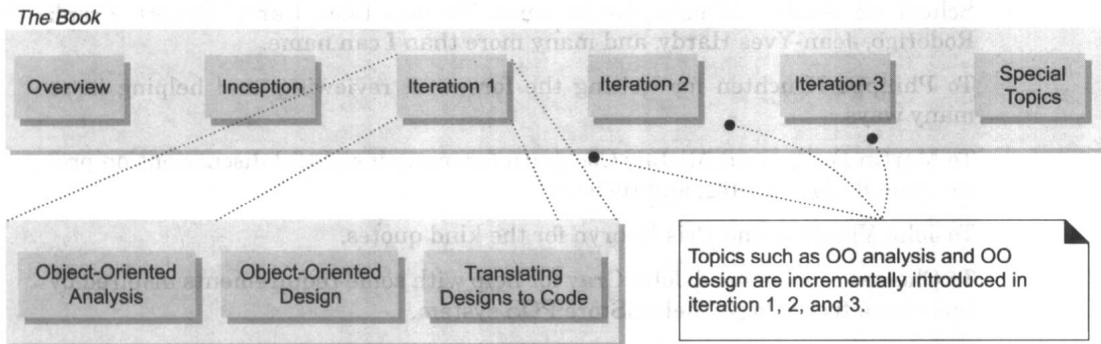


Figure P.1. The organization of the book follows that of a development project.

Web-Related Resources

- Please see www.craiglarman.com for articles related to object technology, patterns, and process.
- Some instructor resources can be found at www.phptr.com/larman.

Enhancements to the First Edition

While retaining the same core as the first edition, the second is refined in many ways, including:

- Use cases are updated to follow the very popular approach of [Cockburn01].
- The well-known Unified Process (UP) is used as the example iterative process within which to introduce OOA/D. Thus, all artifacts are named according to UP terms, such as Domain Model.
- New requirements in the case study, leading to a third iteration.

- Updated treatment of design patterns.
- Introduction to architectural analysis.
- Introduction of Protected Variations as a GRASP pattern.
- A 50/50 balance between sequence and collaboration diagrams.
- The latest UML notation updates.
- Discussion of some practical aspects of drawing using whiteboards or UML CASE tools.

Acknowledgments

First, a very special thanks to my friends and colleagues at Valtech, world-class object developers and iterative development experts, who in some way contributed to, supported, or reviewed the book, including Chris Tarr, Michel Ezran, Tim Snyder, Curtis Hite, Celso Gonzalez, Pascal Roques, Ken DeLong, Brett Schuchert, Ashley Johnson, Chris Jones, Thomas Liou, Darryl Gebert, Frank Rodorigo, Jean-Yves Hardy, and many more than I can name.

To Philippe Kruchten for writing the foreword, reviewing, and helping in so many ways.

To Martin Fowler and Alistair Cockburn for many insightful discussions on process and design, quotes, and reviews.

To John Vlissides and Cris Kobryn for the kind quotes.

To Chelsea Systems and John Gray for help with some requirements inspired by their Java technology ChelseaStore POS system.

To Pete Coad and Dave Astels at TogetherSoft for their support.

Many thanks to the other reviewers, including Steve Adolph, Bruce Anderson, Len Bass, Gary K. Evans, Al Goerner, Luke Hohmann, Eric Lefebvre, David Nunn, and Robert J. White.

Thanks to Paul Becker at Prentice-Hall for believing the first edition would be a worthwhile project, and to Paul Petralia and Patti Guerrieri for shepherding the second.

Finally, a special thanks to Graham Glass for opening a door.

About the Author

Craig Larman serves as Director of Process for Valtech, an international consulting company with divisions in Europe, Asia, and North America, specializing in e-business systems development, object technologies, and iterative development with the Unified Process.

Since the mid 1980s, Craig has helped thousands of developers to apply object-oriented programming, analysis, and design, and assisted organizations adopt iterative development practices.

After a failed career as a wandering street musician, he built systems in APL, PL/I, and CICS in the 1970s. Starting in the early 1980s—after a full recovery—he became interested in artificial intelligence (having little of his own), natural language processing, and knowledge representation, and built knowledge systems with Lisp machines, Lisp, Prolog, and Smalltalk. He plays bad lead guitar in his part-time band, the *Changing Requirements* (it used to be called the *Requirements*, but some band members changed...).

He holds a B.Sc. and M.Sc. in computer science from Simon Fraser University in Vancouver, Canada.

Contact

Craig can be reached at clarman@ieee.org and www.craiglarman.com. He welcomes questions from readers and educators, and speaking, mentoring, and consulting enquiries.

Typographical Conventions

This is a **new term** in a sentence. This is a *Class* or *method* name in a sentence. This is an author reference [Bob67]. A language independent scope resolution operator “--” is used to indicate a class and its associated method as follows: *ClassName--methodName*.

Production Notes

The manuscript of this book was created with Adobe FrameMaker. All drawings were done with Microsoft Visio. The body font is New Century Schoolbook. The final print images were generated as PDF files using Adobe Acrobat Distiller, from PostScript generated by an AGFA driver.

TABLE OF CONTENTS

Foreword xv

Preface xvii

PART I INTRODUCTION

- 1 Object-Oriented Analysis and Design 3
 - Applying UML and Patterns in OOA/D 3
 - Assigning Responsibilities 6
 - What Is Analysis and Design? 6
 - What Is Object-Oriented Analysis and Design? 7
 - An Example 7
 - The UML 10
 - Further Readings 11
- 2 Iterative Development and the Unified Process 13
 - The Most Important UP Idea: Iterative Development 14
 - Additional UP Best Practices and Concepts 18
 - The UP Phases and Schedule-Oriented Terms 19
 - The UP Disciplines (was Workflows) 20
 - Process Customization and the Development Case 23
 - The Agile UP 24
 - The Sequential “Waterfall” Lifecycle 25
 - You Know You Didn’t Understand the UP When... 26
 - Further Readings 26
- 3 Case Study: The NextGen POS System 29
 - The NextGen POS System 29
 - Architectural Layers and Case Study Emphasis 30
 - The Book’s Strategy: Iterative Learning and Development 31

PART II INCEPTION

- 4 Inception 35
 - Inception: An Analogy 36
 - Inception May Be Very Brief 36
 - What Artifacts May Start in Inception? 37
 - You Know You Didn’t Understand Inception When... 38
- 5 Understanding Requirements 41
 - Types of Requirements 42
 - Further Readings 43
- 6 Use-Case Model: Writing Requirements in Context 45
 - Goals and Stories 46
 - Background 46
 - Use Cases and Adding Value 47
 - Use Cases and Functional Requirements 48
 - Use Case Types and Formats 49
 - Fully Dressed Example: Process Sale 50
 - Explaining the Sections 54
 - Goals and Scope of a Use Case 59
 - Finding Primary Actors, Goals, and Use Cases 63
 - Congratulations: Use Cases Have Been Written, and Are Imperfect 67
 - Write Use Cases in an Essential UI-Free Style 68
 - Actors 70
 - Use Case Diagrams 71
 - Requirements in Context and Low-Level Feature Lists 73
 - Use Cases Are Not Object-Oriented 75

	Use Cases Within the UP	75
	Case Study: Use Cases in the NextGen Inception Phase	79
	Further Readings	79
	UP Artifacts and Process Context	81
7	Identifying Other Requirements	83
	NextGen POS Examples	84
	NextGen Example: (Partial) Supplementary Specification	84
	Commentary: Supplementary Specification	88
	NextGen Example: (Partial) Vision	91
	Commentary: Vision	93
	NextGen Example: A (Partial) Glossary	98
	Commentary: Glossary (Data Dictionary)	99
	Reliable Specifications: An Oxymoron?	100
	Online Artifacts at the Project Website	101
	Not Much UML During Inception?	101
	Other Requirement Artifacts Within the UP	101
	Further Readings	104
	UP Artifacts and Process Context	105
8	From Inception to Elaboration	107
	Checkpoint: What Happened in Inception?	108
	On to Elaboration	109
	Planning the Next Iteration	110
	Iteration 1 Requirements and Emphasis: Fundamental OOA/D Skills	112
	What Artifacts May Start in Elaboration?	113
	You Know You Didn't Understand Elaboration When...	114
PART III ELABORATION ITERATION 1		
9	Use-Case Model: Drawing System Sequence Diagrams	117
	System Behavior	118
	System Sequence Diagrams	118
	Example of an SSD	119
	Inter-System SSDs	120
	SSDs and Use Cases	120
	System Events and the System Boundary	120
	Naming System Events and Operations	121
	Showing Use Case Text	122
	SSDs and the Glossary	122
	SSDs Within the UP	123
	Further Readings	124
	UP Artifacts	125
10	Domain Model: Visualizing Concepts	127
	Domain Models	128
	Conceptual Class Identification	132
	Candidate Conceptual Classes for the Sales Domain	136
	Domain Modeling Guidelines	137
	Resolving Similar Conceptual Classes—Register vs. "POST"	139
	Modeling the <i>Unreal</i> World	140
	Specification or Description Conceptual Classes	140
	UML Notation, Models, and Methods: Multiple Perspectives	144
	Lowering the Representational Gap	146
	Example: The NextGen POS Domain Model	148
	Domain Models Within the UP	148
	Further Readings	150

TABLE OF CONTENTS

	UP Artifacts	151
11	Domain Model: Adding Associations	153
	Associations	153
	The UML Association Notation	154
	Finding Associations—Common Associations List	155
	Association Guidelines	157
	Roles	157
	How Detailed Should Associations Be?	159
	Naming Associations	160
	Multiple Associations Between Two Types	161
	Associations and Implementation	161
	NextGen POS Domain Model Associations	162
	NextGen POS Domain Model	163
12	Domain Model: Adding Attributes	167
	Attributes	167
	UML Attribute Notation	168
	Valid Attribute Types	168
	Non-primitive Data Type Classes	170
	Design Creep: No Attributes as Foreign Keys	172
	Modeling Attribute Quantities and Units	173
	Attributes in the NextGen Domain Model	174
	Multiplicity From SalesLineItem to Item	175
	Domain Model Conclusion	175
13	Use-Case Model: Adding Detail with Operation Contracts	177
	Contracts	177
	Example Contract: enterItem	178
	Contract Sections	179
	Postconditions	179
	Discussion—enterItem Postconditions	182
	Writing Contracts Leads to Domain Model Updates	183
	When Are Contracts Useful? Contracts vs. Use Cases?	183
	Guidelines: Contracts	184
	NextGen POS Example: Contracts	185
	Changes to the Domain Model	186
	Contracts, Operations, and the UML	186
	Operation Contracts Within the UP	188
	Further Readings	191
14	From Requirements to Design in this Iteration	193
	Iteratively Do the Right Thing, Do the Thing Right	193
	Didn't That Take Weeks To Do? No, Not Exactly.	194
	On to Object Design	194
15	Interaction Diagram Notation	197
	Sequence and Collaboration Diagrams	198
	Example Collaboration Diagram: makePayment	199
	Example Sequence Diagram: makePayment	200
	Interaction Diagrams Are Valuable	200
	Common Interaction Diagram Notation	201
	Basic Collaboration Diagram Notation	202
	Basic Sequence Diagram Notation	208
16	GRASP: Designing Objects with Responsibilities	215
	Responsibilities and Methods	216
	Responsibilities and Interaction Diagrams	217
	Patterns	218

TABLE OF CONTENTS

	GRASP: Patterns of General Principles in Assigning Responsibilities	219
	The UML Class Diagram Notation	220
	Information Expert (or Expert)	221
	Creator	226
	Low Coupling	229
	High Cohesion	232
	Controller	237
	Object Design and CRC Cards	245
	Further Readings	246
17	Design Model: Use-Case Realizations with GRASP Patterns	247
	Use-Case Realizations	248
	Artifact Comments	249
	Use-Case Realizations for the NextGen Iteration	252
	Object Design: makeNewSale	253
	Object Design: enterItem	255
	Object Design: endSale	260
	Object Design: makePayment	264
	Object Design: startUp	269
	Connecting the UI Layer to the Domain Layer	273
	Use-Case Realizations Within the UP	276
	Summary	278
18	Design Model: Determining Visibility	279
	Visibility Between Objects	279
	Visibility	280
	Illustrating Visibility in the UML	284
19	Design Model: Creating Design Class Diagrams	285
	When to Create DCDs	285
	Example DCD	286
	DCD and UP Terminology	286
	Domain Model vs. Design Model Classes	287
	Creating a NextGen POS DCD	287
	Notation for Member Details	296
	DCDs, Drawing, and CASE Tools	298
	DCDs Within the UP	298
	UP Artifacts	299
20	Implementation Model: Mapping Designs to Code	301
	Programming and the Development Process	302
	Mapping Designs to Code	304
	Creating Class Definitions from DCDs	304
	Creating Methods from Interaction Diagrams	307
	Container/Collection Classes in Code	309
	Exceptions and Error Handling	309
	Defining the Sale--makeLineItem Method	310
	Order of Implementation	311
	Test-First Programming	311
	Summary of Mapping Designs to Code	313
	Introduction to the Program Solution	313

PART IV ELABORATION ITERATION 2

21	Iteration 2 and its Requirements	319
	Iteration 2 Emphasis: Object Design and Patterns	319
	From Iteration 1 to 2	319
	Iteration 2 Requirements	321

	Refinement of Analysis-oriented Artifacts in this Iteration	322
22	GRASP: More Patterns for Assigning Responsibilities	325
	Polymorphism	326
	Pure Fabrication	329
	Indirection	332
	Protected Variations	334
23	Designing Use-Case Realizations with GoF Design Patterns	341
	Adapter (GoF)	342
	“Analysis” Discoveries During Design: Domain Model	345
	Factory (GoF)	346
	Singleton (GoF)	348
	Conclusion of the External Services with Varying Interfaces Problem	352
	Strategy (GoF)	353
	Composite (GoF) and Other Design Principles	358
	Facade (GoF)	368
	Observer/Publish-Subscribe/Delegation Event Model (GoF)	372
	Conclusion	380
	Further Readings	380
PART V ELABORATION ITERATION 3		
24	Iteration 3 and Its Requirements	383
	Iteration 3 Requirements	383
	Iteration 3 Emphasis	383
25	Relating Use Cases	385
	The include Relationship	386
	Terminology: Concrete, Abstract, Base, and Addition Use Cases	388
	The extend Relationship	389
	The generalize Relationship	390
	Use Case Diagrams	391
26	Modeling Generalization	393
	New Concepts for the Domain Model	393
	Generalization	396
	Defining Conceptual Superclasses and Subclasses	397
	When to Define a Conceptual Subclass	400
	When to Define a Conceptual Superclass	403
	NextGen POS Conceptual Class Hierarchies	403
	Abstract Conceptual Classes	406
	Modeling Changing States	408
	Class Hierarchies and Inheritance in Software	409
27	Refining the Domain Model	411
	Association Classes	411
	Aggregation and Composition	414
	Time Intervals and Product Prices--Fixing an Iteration 1 “Error”	418
	Association Role Names	419
	Roles as Concepts vs. Roles in Associations	420
	Derived Elements	421
	Qualified Associations	422
	Reflexive Associations	423
	Ordered Elements	423
	Using Packages to Organize the Domain Model	423
28	Adding New SSDs and Contracts	431
	New System Sequence Diagrams	431
	New System Operations	433
	New System Operation Contracts	434