

APPLYING UML AND PATTERNS

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

SECOND EDITION

Free Book for
Everyone

"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 unreserved choice."

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

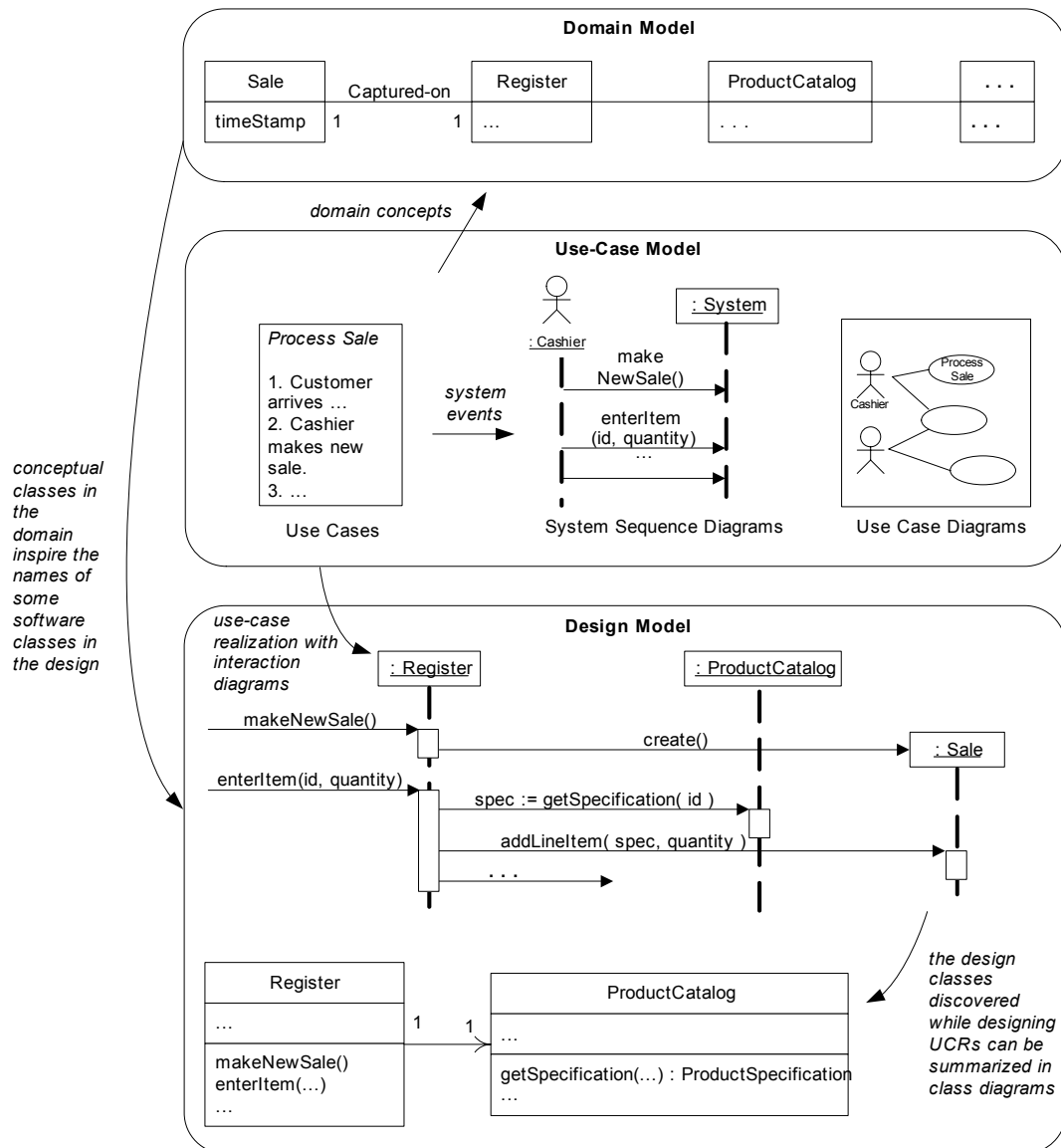
CRAIG LARMAN

Foreword by Philippe Kruchten

Sample Unified Process Artifacts and Timing (s-start; r-refine)

Discipline	Artifact Iteration [^]	Incep. I1	Elab. EL.En	Const. CL.Cn	Trans. T1..T2
Business Modeling	Domain Model		s		
Requirements	Use-Case Model	s	r		
	Vision	a	r		
	Supplementary Specification	s	r		
	Glossary	s	r		
Design	Design Model SW Architecture Document Data Model		s s s	r r	
Implementation	Implementation Model		s	r	r
Project Management	t SW Development Plan	s	r	r	r
Testing	Test Model		s	r	
Environment	Development Case	s	r		

Sample Unified Process Artifact Relationships



General Responsibility Assignment Software Patterns (GRASP)

Pattern	Description
Information Expert	A general principle of object design and responsibility assignment? Assign a responsibility to the information expert — the class that has the information necessary to fulfill the responsibility.
Creator	Who creates? (Note that Factory is a common alternate solution.) Assign class B the responsibility to create an instance of class A if one of these is true: 1. B contains A 2. B aggregates A 3. B has the initializing data for A 4. B records A 5. B closely uses A
Controller	Who handles a system event? Assign the responsibility for handling a system event message to a class representing one of these choices: 1. Represents the overall system, device, or a subsystem (facade controller). 2. Represents a use case scenario within which the system event occurs (use-case or session controller)
Low Coupling (evaluative)	How to support low dependency and increased reuse? Assign responsibilities so that (unnecessary) coupling remains low.
High Cohesion (evaluative)	How to keep complexity manageable? Assign responsibilities so that cohesion remains high.
Polymorphism	Who is responsible when behavior varies by type? When related alternatives or behaviors vary by type (class), assign responsibility for the behavior — using polymorphic operations — to the types for which the behavior varies.
Pure Fabrication	Who is responsible when you are desperate, and do not want to violate high cohesion and low coupling? Assign a highly cohesive set of responsibilities to an artificial or convenience "behavior" class that does not represent a problem domain concept — something made up, in order to support high cohesion, low coupling, and reuse.
Indirection	How to assign responsibilities to avoid direct coupling? Assign the responsibility to an intermediate object to mediate between other components or services, so that they are not directly coupled.
Protected Variations	How to assign responsibilities to objects, subsystems, and systems so that the variations or instability in these elements do not have an undesirable impact on other elements? Identify points of predicted variation or instability; assign responsibilities to create a stable "interface" around them.

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 <i>Case</i>	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 <i>Case</i> Diagrams	71
	Requirements in Context and Low-Level Feature Lists	73
	Use Cases Are Not Object-Oriented	75

TABLE OF CONTENTS

	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?	118
	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 <i>in</i> 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	2.52
	Object Design: makeNewSale	253
	Object Design: enter-Item	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 BCD	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

TABLE OF CONTENTS

	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

TABLE OF CONTENTS

29	Modeling Behavior in Statechart Diagrams	437
	Events, States, and Transitions	437
	Statechart Diagrams	438
	Statechart Diagrams in the UP?	439
	Use Case Statechart Diagrams	439
	Use Case Statechart Diagrams for the POS Application	441
	Classes that Benefit from Statechart Diagrams	441
	Illustrating External and Interval Events	443
	Additional Statechart Diagram Notation	444
	Further Readings	446
30	Designing the Logical Architecture with Patterns	447
	Software Architecture	448
	Architectural Pattern: Layers	450
	The Model-View Separation Principle	471
	Further Readings	474
31	Organizing the Design and Implementation Model Packages	475
	Package Organization Guidelines	476
	More UML Package Notation	482
	Further Readings	483
32	Introduction to Architectural Analysis and the SAD	485
	Architectural Analysis	486
	Types and Views of Architecture	488
	The Science: Identification and Analysis of Architectural Factors	488
	Example: Partial NextGen POS Architectural Factor Table	491
	The Art: Resolution of Architectural Factors	493
	Summary of Themes in Architectural Analysis	499
	Architectural Analysis within the UP	500
	Further Readings	505
33	Designing More Use-Case Realizations with Objects and Patterns	
507		
	Failover to Local Services; Performance with Local Caching	507
	Handling Failure	512
	Failover to Local Services with a Proxy (GoF)	519
	Designing for Non-Functional or Quality Requirements	523
	Accessing External Physical Devices with Adapters; Buy vs. Build	523
	Abstract Factory (GoF) for Families of Related Objects	525
	Handling Payments with Polymorphism and Do It Myself	528
	Conclusion	535
34	Designing a Persistence Framework with Patterns	537
	The Problem: Persistent Objects	538
	The Solution: A Persistence Service from a Persistence Framework	538
	Frameworks	539
	Requirements for the Persistence Service and Framework	540
	Key Ideas	540
	Pattern: Representing Objects as Tables	541
	UML Data Modeling Profile	541
	Pattern: Object Identifier	542
	Accessing a Persistence Service with a Facade	543
	Mapping Objects: Database Mapper or Database Broker Pattern	543
	Framework Design with the Template Method Pattern	546
	Materialization with the Template Method Pattern	546
	Configuring Mappers with a MapperFactory	552
	Pattern: Cache Management	552
	Consolidating and Hiding SQL Statements in One Class	553

TABLE OF CONTENTS

29	Modeling Behavior in Statechart Diagrams	437
	Events, States, and Transitions	437
	Statechart Diagrams	438
	Statechart Diagrams in the UP?	439
	Use Case Statechart Diagrams	439
	Use Case Statechart Diagrams for the POS Application	441
	Classes that Benefit from Statechart Diagrams	441
	Illustrating External and Interval Events	443
	Additional Statechart Diagram Notation	444
	Further Readings	446
30	Designing the Logical Architecture with Patterns	447
	Software Architecture	448
	Architectural Pattern: Layers	450
	The Model-View Separation Principle	471
	Further Readings	474
31	Organizing the Design and Implementation Model Packages	475
	Package Organization Guidelines	476
	More UML Package Notation	482
	Further Readings	483
32	Introduction to Architectural Analysis and the SAD	485
	Architectural Analysis	486
	Types and Views of Architecture	488
	The Science: Identification and Analysis of Architectural Factors	488
	Example: Partial NextGen POS Architectural Factor Table	491
	The Art: Resolution of Architectural Factors	493
	Summary of Themes in Architectural Analysis	499
	Architectural Analysis within the UP	500
	Further Readings	505
33	Designing More Use-Case Realizations with Objects and Patterns	
507		
	Failover to Local Services; Performance with Local Caching	507
	Handling Failure	512
	Failover to Local Services with a Proxy (GoF)	519
	Designing for Non-Functional or Quality Requirements	523
	Accessing External Physical Devices with Adapters; Buy vs. Build	523
	Abstract Factory (GoF) for Families of Related Objects	525
	Handling Payments with Polymorphism and Do It Myself	528
	Conclusion	535
34	Designing a Persistence Framework with Patterns	537
	The Problem: Persistent Objects	538
	The Solution: A Persistence Service from a Persistence Framework	538
	Frameworks	539
	Requirements for the Persistence Service and Framework	540
	Key Ideas	540
	Pattern: Representing Objects as Tables	541
	UML Data Modeling Profile	541
	Pattern: Object Identifier	542
	Accessing a Persistence Service with a Facade	543
	Mapping Objects: Database Mapper or Database Broker Pattern	543
	Framework Design with the Template Method Pattern	546
	Materialization with the Template Method Pattern	546
	Configuring Mappers with a MapperFactory	552
	Pattern: Cache Management	552
	Consolidating and Hiding SQL Statements in One Class	553

TABLE OF CONTENTS

Transactional States and the State Pattern	554	Designing a Transaction with the Command Pattern	556	Lazy Materialization with a Virtual Proxy	559	How to Represent Relationships in Tables	562	PersistentObject Superclass and Separation of Concerns	563	Unresolved Issues	564
--	-----	--	-----	---	-----	--	-----	--	-----	-------------------	-----

PART VI SPECIAL TOPICS

35	On Drawing and Tools	567	On Speculative Design and Visual Thinking	567	Suggestions for UML Drawing Within the Development Process	568	Tools and Sample Features	571	Example Two	573
36	Introduction to Iterative Planning and Project Issues	575	Ranking Requirements	576	Ranking Project Risks	579	Adaptive vs. Predictive Planning	579	Phase and Iteration Plans	581
	Iteration Plan: What to Do in the Next Iteration?	582	Requirements Tracking Across Iterations	583	The (Invalidity of Early Estimates	585	Organizing Project Artifacts	585	Some Team Iteration Scheduling Issues	586
	You Know You Didn't Understand Planning in the UP When...	588	Further Readings	588						
37	Comments on Iterative Development and the UP	589	Additional UP Best Practices and Concepts	589	The Construction and Transition Phases	591	Other Interesting Practices	592	Motivations for Timeboxing an Iteration	593
	The Sequential "Waterfall" Lifecycle	593	Usability Engineering and User Interface Design	599	The UP Analysis Model	599	The RUP Product	600	The Challenge and Myths of Reuse	601
38	More UML Notation	603	General Notation	603	Implementation Diagrams	604	Template (Parameterized, Generic) Class	606	Activity Diagrams	607
	Bibliography	609	Glossary	615	Index	621				

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

fled 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 OOA/D. 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.I).

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.

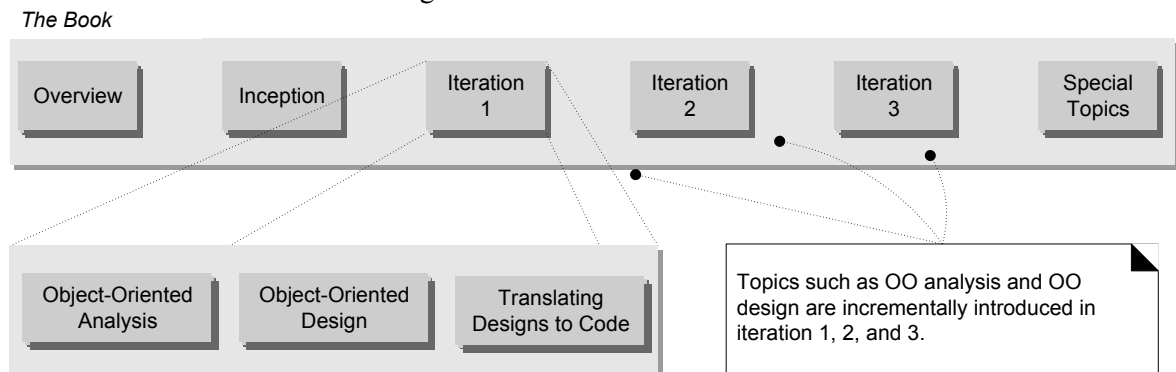


Figure P.I. The organization of the book follows that of a development project.

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.

PREFACE

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 Roderigo, 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 Goad 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.

PREFACE

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.

Craig can be reached at clarman@acm.org and www.craiglarman.com.

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.