

# Use Case Driven Object Modeling with UML

Theory and Practice



Doug Rosenberg and  
Matt Stephens

## **Use Case Driven Object Modeling with UML: Theory and Practice**

**Copyright © 2007 by Doug Rosenberg and Matt Stephens**

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-59059-774-3

ISBN-10 (pbk): 1-59059-774-5

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Lead Editor: Jonathan Gennick

Technical Reviewer: Dr. Charles Suscheck

Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick,

Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Matt Wade

Senior Project Manager: Tracy Brown Collins

Copy Edit Manager: Nicole Flores

Assistant Production Director: Kari Brooks-Copony

Senior Production Editor: Laura Cheu

Compositor: Linda Weidemann, Wolf Creek Press

Proofreader: Nancy Riddiough

Indexer: Toma Mulligan

Artist: Kinetic Publishing Services, LLC

Cover Designer: Kurt Krames

Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail [orders-ny@springer-sbm.com](mailto:orders-ny@springer-sbm.com), or visit <http://www.springeronline.com>.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail [info@apress.com](mailto:info@apress.com), or visit <http://www.apress.com>.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The UML model and source code for the example use cases in this book are available to readers at <http://www.apress.com> and <http://www.iconixprocess.com/InternetBookstore>.

*For Rob, who has the brightest future of anyone I know.  
Keep locating your fastball in unhittable spots,  
and good things will continue to happen.*

*—Doug Rosenberg*

*To Michelle, for her never-ending patience and support.*

*—Matt*

# Contents at a Glance

About the Authors .....	xv
About the Technical Reviewer .....	xvii
Acknowledgments .....	xix
Preface .....	xxi
Introduction .....	xxvii
<b>CHAPTER 1</b> Introduction to ICONIX Process .....	1
<b>PART 1</b> ■ ■ ■ Requirements Definition	
<b>CHAPTER 2</b> Domain Modeling .....	23
<b>CHAPTER 3</b> Use Case Modeling .....	49
<b>CHAPTER 4</b> Requirements Review .....	83
<b>PART 2</b> ■ ■ ■ Analysis, Conceptual Design, and Technical Architecture	
<b>CHAPTER 5</b> Robustness Analysis .....	101
<b>CHAPTER 6</b> Preliminary Design Review .....	143
<b>CHAPTER 7</b> Technical Architecture .....	159
<b>PART 3</b> ■ ■ ■ Design and Coding	
<b>CHAPTER 8</b> Sequence Diagrams .....	185
<b>CHAPTER 9</b> Critical Design Review .....	233
<b>CHAPTER 10</b> Implementation: Getting from Detailed Design to Code .....	257
<b>CHAPTER 11</b> Code Review and Model Update .....	297

## PART 4 ■ ■ ■ Testing and Requirements Traceability

■ CHAPTER 12	Design-Driven Testing .....	329
■ CHAPTER 13	Addressing Requirements.....	373

## PART 5 ■ ■ ■ Appendixes

■ APPENDIX A	What's New in UML 2.0 .....	395
■ APPENDIX B	Spring Bin.....	409
■ INDEX	.....	425

# Contents

About the Authors .....	xv
About the Technical Reviewer .....	xvii
Acknowledgments .....	xix
Preface .....	xxi
Introduction .....	xxvii

<b>CHAPTER 1 Introduction to ICONIX Process .....</b>	<b>1</b>
ICONIX Process in Theory .....	2
Overview: Getting from Use Cases to Source Code .....	2
Requirements .....	4
Analysis/Preliminary Design .....	9
Detailed Design .....	12
Implementation .....	15
Extensions to ICONIX Process .....	19
Persona Analysis .....	19
Test-Driven Development (TDD) .....	19
Driving Test Cases from the Analysis Model .....	20
ICONIX Process in Practice: The Internet Bookstore Example .....	20
Summary .....	20

## PART 1 ■ ■ ■ Requirements Definition

<b>CHAPTER 2 Domain Modeling .....</b>	<b>23</b>
The 10,000-Foot View .....	24
What's a Domain Model? .....	24
Why Start with the Domain Model Instead of Use Cases? .....	25
Domain Modeling in Theory .....	26
Top 10 Domain Modeling Guidelines .....	26
Internet Bookstore: Extracting the First-Pass Domain Model from High-Level Requirements .....	30
Internet Bookstore: Second Attempt at the Domain Model .....	35
Internet Bookstore: Building Generalization Relationships .....	37

Domain Modeling in Practice . . . . .	39
Exercises . . . . .	39
More Practice . . . . .	45
Summary . . . . .	47
<b>CHAPTER 3 Use Case Modeling . . . . .</b>	<b>49</b>
The 10,000-Foot View . . . . .	49
Why Do I Need Use Cases in Addition to Functional Requirements? . . . . .	50
Don't Forget the Rainy-Day Scenarios . . . . .	50
Do an Initial Domain Model Before You Write the Use Cases . . . . .	50
Driving Your Design (and Your Tests) from the Use Cases . . . . .	51
Use Case Modeling in Theory . . . . .	51
Top 10 Use Case Modeling Guidelines . . . . .	51
Organizing Use Cases into Packages: Internet Bookstore . . . . .	61
Use Case Relationship Roundup . . . . .	67
Internet Bookstore: Refining Use Cases . . . . .	70
Internet Bookstore: Basic and Alternate Courses . . . . .	72
A Couple of Thoughts on Use Case Templates . . . . .	74
Use Case or Algorithm? . . . . .	76
Use Case Modeling in Practice . . . . .	77
Exercises . . . . .	77
Exercise Solutions . . . . .	78
More Practice . . . . .	80
Summary . . . . .	81
<b>CHAPTER 4 Requirements Review . . . . .</b>	<b>83</b>
Requirements Review in Theory . . . . .	84
Why Review Requirements? . . . . .	84
Top 10 Requirements Review Guidelines . . . . .	85
Allocating Functional Requirements to Use Cases . . . . .	89
Requirements Review in Practice: Internet Bookstore . . . . .	89
Removing Everything That's Out of Scope . . . . .	90
Naming Participating Domain Objects . . . . .	92
Making Sure You Have All the Alternate Courses . . . . .	93
Checking That the Use Case Text Isn't Too Abstract . . . . .	93
Changing Passive Voice to Active Voice . . . . .	95
Tracing Each Requirement to Its Use Cases . . . . .	96
Summary . . . . .	97

## PART 2 ■ ■ ■ Analysis, Conceptual Design, and Technical Architecture

<b>CHAPTER 5</b>	<b>Robustness Analysis</b> .....	101
	The 10,000-Foot View .....	101
	Where Does Robustness Analysis Fit into the Process? .....	102
	Like Learning to Ride a Bicycle .....	102
	Anatomy of a Robustness Diagram .....	103
	Robustness Analysis in Theory .....	104
	Top 10 Robustness Analysis Guidelines .....	104
	More About Robustness Diagram Rules .....	112
	How Do You Perform Robustness Analysis? .....	114
	Updating Your Domain (Static) Model .....	125
	Robustness Analysis in Practice .....	128
	Exercises .....	128
	Exercise Solutions .....	132
	More Practice .....	140
	Summary .....	141
<b>CHAPTER 6</b>	<b>Preliminary Design Review</b> .....	143
	Preliminary Design Review in Theory .....	144
	Why Do a PDR At All? .....	144
	Top 10 PDR Guidelines .....	145
	Preliminary Design Review in Practice: Internet Bookstore .....	149
	PDR for the “Write Customer Review” Robustness Diagram .....	149
	The Finished “Write Customer Review” Robustness Diagram .....	155
	Summary .....	157
<b>CHAPTER 7</b>	<b>Technical Architecture</b> .....	159
	The 10,000-Foot View .....	160
	What Is Technical Architecture? .....	160
	What Are the Duties of a Technical Architect? .....	160
	Technical Architecture in Theory .....	161
	Top 10 Technical Architecture Guidelines .....	161
	Architectural Layering .....	162
	Technical Architecture in Practice: Internet Bookstore .....	164
	About Spring Framework .....	164
	Anatomy of Spring Framework .....	165



The Internet Bookstore Architecture . . . . .	172
Layered Architecture . . . . .	172
Flow of Events . . . . .	178
Testability . . . . .	179
Web Security . . . . .	179
Top 10 Technical Architecture Errors (the “Don’ts”) . . . . .	180
Summary . . . . .	181

## PART 3 ■ ■ ■ Design and Coding

<b>CHAPTER 8</b> <b>Sequence Diagrams</b> . . . . .	185
The 10,000-Foot View . . . . .	185
Sequence Diagrams and Detailed OOD . . . . .	186
Sequence Diagram Notation . . . . .	186
Sequence Diagramming in Theory . . . . .	187
Top 10 Sequence Diagramming Guidelines . . . . .	187
How to Draw a Sequence Diagram: Four Essential Steps . . . . .	195
Continuing the Internet Bookstore Example . . . . .	206
Updating Your Class Diagrams As You Go Along . . . . .	210
Synchronizing the Static and Dynamic Parts of the Model . . . . .	211
Internet Bookstore: Updating the Static Model . . . . .	211
Sequence Diagramming in Practice . . . . .	217
Exercises . . . . .	217
Exercise Solutions . . . . .	221
More Practice . . . . .	228
Summary . . . . .	230
 <b>CHAPTER 9</b> <b>Critical Design Review</b> . . . . .	 233
The 10,000-Foot View . . . . .	234
Critical Design Review in Theory . . . . .	235
Top 10 Critical Design Review Guidelines . . . . .	235
Using the Class Diagrams to Find Errors on the Sequence Diagrams . . . . .	 238

Critical Design Review in Practice: Internet Bookstore . . . . . 238  
 CDR for the “Show Book Details” Use Case . . . . . 238  
 CDR for the “Write Customer Review” Use Case . . . . . 245  
 The Updated Bookstore Diagrams . . . . . 252  
 Summary . . . . . 255

**CHAPTER 10 Implementation: Getting from Detailed Design to Code** . . . . . 257

The 10,000-Foot View . . . . . 258  
 Programmer-Driven Design . . . . . 258  
 Spring Framework . . . . . 258  
 Implementation in Theory: Getting from Design to Code . . . . . 258  
 Top 10 Implementation Guidelines . . . . . 259  
 Implementation in Practice: Internet Bookstore . . . . . 263  
 Creating the Database . . . . . 263  
 Preparing the Style Sheet . . . . . 265  
 Mapping Domain (Entity) Classes to Real Classes . . . . . 266  
 Implementing the “Show Book Details” Use Case . . . . . 268  
 Implementing the “Write Customer Review” Use Case . . . . . 278  
 More Practice . . . . . 294  
 Summary . . . . . 295

**CHAPTER 11 Code Review and Model Update** . . . . . 297

The 10,000-Foot View . . . . . 298  
 Code Review and Model Update in Theory . . . . . 298  
 Top 10 Code Review and Model Update Guidelines . . . . . 299  
 Why Are Code Reviews Necessary After All That Design Work? . . . . . 302  
 Code Review and Model Update in Practice . . . . . 303  
 Code Review and Model Update Checklist . . . . . 304  
 “Show Book Details” Use Case . . . . . 304  
 “Write Customer Review” Use Case . . . . . 309  
 Future Iterations . . . . . 324  
 Summary . . . . . 325

## PART 4 ■ ■ ■ Testing and Requirements Traceability

■ CHAPTER 12	<b>Design-Driven Testing</b> .....	329
	Design-Driven Testing in Theory .....	330
	Top 10 Design-Driven Testing Guidelines .....	330
	Different Kinds of Testing .....	331
	Driving Test Cases from Robustness Diagrams .....	334
	Using the Agile ICONIX/EA Add-in .....	336
	Driving Unit Tests from the Test Cases .....	338
	A Quick Introduction to JUnit .....	339
	Writing Effective Unit Tests .....	342
	Design-Driven Testing in Practice .....	343
	Unit Tests for the Internet Bookstore .....	344
	Top 10 Design-Driven Testing Errors (the “Don’ts”) .....	369
	More Practice .....	370
	Summary .....	371
■ CHAPTER 13	<b>Addressing Requirements</b> .....	373
	Requirements Gathering in Theory .....	374
	Top 10 Requirements Gathering Guidelines .....	374
	Why Bother Tracking Requirements? .....	377
	Requirements Allocation and Traceability in Theory .....	378
	Requirements Gathering in Practice .....	379
	Organizing Requirements in EA: BillyBob 2.0 .....	379
	Using a Visual Modeling Tool to Support Requirements .....	382
	More Practice .....	389
	Summary .....	390

## PART 5 ■ ■ ■ Appendixes

■ <b>APPENDIX A</b>	<b>What's New in UML 2.0</b> .....	395
	Overview of Changes in UML 2.0 .....	395
	Composite Structure Diagrams .....	396
	Activity and State Diagrams .....	399
	Sequence and Interaction Overview Diagrams .....	401
	Timing Diagrams .....	404
	Component and Deployment Diagrams .....	406
	What's Still Missing in UML .....	407
■ <b>APPENDIX B</b>	<b>Spring Bin</b> .....	409
	Spring in More Detail .....	409
	A (Very) Brief Example of IoC .....	409
	Models, Views, and Controllers .....	412
	Internet Bookstore Design: Spring Details .....	414
	"Show Book Details" Use Case .....	414
	"Write Customer Review" Use Case .....	416
	Internet Bookstore Implementation: Spring Details .....	417
	Folder Structure .....	418
	Contents of the war\WEB-INF Folder .....	418
	Contents of the war\WEB-INF\jsp and war\WEB-INF\jsp\include Folders .....	421
	Java Package Hierarchy .....	422
■ <b>INDEX</b>	.....	425

# About the Authors



■ **DOUG ROSENBERG** is the founder and president of ICONIX Software Engineering, Inc. ([www.iconixsw.com](http://www.iconixsw.com)). Doug spent the first 15 years of his career writing code for a living before moving on to managing programmers, developing software design tools, and teaching object-oriented analysis and design.

Doug has been providing system development tools and training for nearly two decades, with particular emphasis on object-oriented methods. He developed a unified Booch/Rumbaugh/Jacobson design method in 1993 that preceded Rational's UML by several years. He has produced more than a dozen multimedia tutorials on object technology, including "COMPREHENSIVE COM" and "Enterprise Architect for Power Users," and is the coauthor of *Use Case Driven Object Modeling with UML* (Addison-Wesley, 1999) and *Applying Use Case Driven Object Modeling with UML* (Addison-Wesley, 2001), both with Kendall Scott, as well as *Extreme Programming Refactored: The Case Against XP* (Apress, 2003) with Matt Stephens, and *Agile Development with ICONIX Process* (Apress, 2005) with Matt Stephens and Mark Collins-Cope.

A few years ago, Doug started a second business, an online travel website ([www.VResorts.com](http://www.VResorts.com)) that features his virtual reality photography and some innovative mapping software.



■ **MATT STEPHENS** is a Java developer, project leader, and technical architect based in Central London. He's been developing software commercially for over 15 years, and has led many agile projects through successive customer releases. He has spoken at a number of software conferences on OO development topics, and his work appears regularly in a variety of software journals.

Matt is the coauthor of *Extreme Programming Refactored: The Case Against XP* (Apress, 2003) with Doug Rosenberg, and *Agile Development with ICONIX Process* (Apress, 2005) with Doug Rosenberg and Mark Collins-Cope.

Catch Matt online at [www.softwarereality.com](http://www.softwarereality.com).

# About the Technical Reviewer

■ **DR. CHARLES SUSCHECK** is an assistant professor of computer information systems at Colorado State University, Pueblo campus. He specializes in software development methodologies and project management, and has over 20 years of professional experience in information technology.

Dr. Suscheck has held the positions of process architect, director of research, principal consultant, and professional trainer at some of the most recognized companies in America. He has spoken at national and international conferences on topics related to project management. Most recently, he's been heavily involved in delivering the "ICONIX Process Roadmap" (as defined by the activity diagrams in this book) via the Eclipse Process Framework.

# Acknowledgments

**F**irst and foremost, **thanks to Gary Cornell** for picking up this project midstream.

Thanks to Geoff Sparks and the folks at Sparx Systems for building a great product, for tailoring it to support ICONIX Process, and for helping us with the UML 2.0 tutorial in Appendix A.

Thanks to Philip Nortey for his valuable feedback and his contribution to the chapter on design-driven testing; to Chuck Suscheck for his reviews and insights, especially about the student exercises; and to Mark Collins-Cope for his contribution to the architecture chapter.

And thanks, of course, to the Apress team: Gary; our editor, Jonathan Gennick; “The PM,” Tracy Brown-Collins (Queen of the 48-hour chapter-editing turnaround deadline), without whose schedule this project would have forever remained in “manuscript paralysis”; “The World’s Greatest Copy Editor” (once again), Nicole Flores; Diana Van Winkle for the outstanding design; and our production editor, Laura Cheu.

# Preface

## Matt's Preface

This book illustrates how to get from use cases to working, maintainable source code in as few steps as possible . . . but without cutting the essential corners. It's also about how to minimize the amount of rework you need to do once you've *gotten* to source code.

## Learning by Doing

In this book we've tried to capture the essential qualities of Doug's ICONIX training courses—that is, the “magic qualities” of **learning by doing**. The ICONIX Jumpstart courses are very practical and hands-on; they draw students in by encouraging them to learn new skills by practicing, often on the real projects that they'll be returning to once the course is finished.

This idea of learning by doing has long been recognized as an optimal form of education. Even at the start of the twentieth century, John Dewey, an American psychologist and educational reformer, recognized that learning from experience gives rise to increasing productivity. The key is to engage the brain with practical tasks rather than to fall into the all-too-familiar “study trap” of rote learning. Memorizing long lists of names or API functions might help someone score highly on a test, but it isn't the same as understanding a subject in depth. For one thing, people tend not to retain information for very long if they've simply memorized it.

In this book, we do several things to avoid the “rote learning” trap. We walk through example diagrams, each starting with a blank screen, and show the steps—and, essentially, the thought process—involved in creating the various types of diagrams. Each step in the ICONIX Process finishes with a review. For the review milestones, we've had some fun and created fictional dialogues between a reviewer and a developer, to demonstrate the sorts of issues that reviewers or senior developers should address at each stage. We also highlight the most common (and the most dangerous) mistakes that developers tend to make.

A key part of learning by doing concerns **learning from your mistakes**. From the day we're born, we learn by discovering how *not* to do things, and then trying over and over until we get it right. Experts eventually “perfect” their art because they no longer make mistakes (at least none that they'll admit to!). So again, we've applied the principle in this book and created an Internet Bookstore example that we follow from use cases to source code, making plenty of “deliberate mistakes” along the way, which then get corrected. Also, throughout the book, you'll find **workbook exercises**, **student exercises**, and **inline exercises** within the chapters. (See the “Introduction” section for more information about these different types of exercises.)

The large number of exercises and step-by-step examples should help to explain why this book contains around 400 pages, to describe a process that is essentially “minimal yet sufficient.” You could say that it's a 150-page book at heart, but it's packed with an unusual number of exercises and examples. It's safe to say that after reading this book and completing all the exercises, you'll have a thorough, in-depth understanding of use case-driven object modeling!



## ICONIX: A Pluggable Process

ICONIX Process is a “cookbook” process in that it describes a series of specific steps that we’ve found work really well on many different projects. However, it doesn’t prescribe the project life-cycle side of things in the way that most other development methodologies do.

So the decision of whether to do just a little bit of up-front modeling before code (one use case at a time) or model all the use cases first before coding is entirely yours to make. You can be as agile (with short iterations and quick, successive releases) or as “waterfall” (first writing all the requirements, then doing all the design, and then writing all the code) as befits your project, and still be following ICONIX Process.<sup>1</sup>

For this reason, the process should plug neatly into other development methodologies, as it covers the analysis and design steps but doesn’t make any fixed assumptions about the project life cycle. But however you choose to apply the process to your own projects, we hope you’ll start to see positive results very quickly.

Matt Stephens

Software Reality, [www.softwarereality.com](http://www.softwarereality.com)

## Doug’s Preface

It was 13 or 14 years ago, somewhere around 1992 or 1993, when one of my first training clients, Dan Mosten of Philip Morris in New York, said to me, “You should write a cookbook on how to design for OO. My people like cookbooks.”

At that time, Grady Booch was at Rational, Jim Rumbaugh was at GE writing books about OMT, and Ivar Jacobson was in Sweden working on his Objectory CASE Tool. There was no UML, no Java language, no C#.NET, and the Internet itself largely existed only in universities. Smalltalk and C++ were the dominant object-oriented (OO) languages. The ancestor of Rational Rose was being developed by Jon Hopkins at Palladio Software as a Booch diagramming tool for the PC. There was no eXtreme Programming (jumping too quickly to code was known as “hacking” back then), and no Agile Manifesto had yet declared tools and process to be second-class citizens.

## The More Things Change, the More They Stay the Same

At ICONIX, we were trying to make some sense out of OO analysis and design (like everybody else), and our efforts produced a tool called ObjectModeler, which supported Booch, Rumbaugh (OMT), and Jacobson (Objectory) methods. We got into training because we had to—nobody would buy our object-oriented analysis and design (OOAD) tool if they didn’t understand OOAD.

We synthesized what is now known as ICONIX Process (and was originally called “A Unified Object Modeling Approach”) from what we felt were the best aspects of the three methodologies that were combined a few years later to form the UML. As we did this, ***it seemed clear that the art of driving object models from use cases ought to be the core of our approach***, and

---

1. Most projects benefit from being somewhere between these two extremes. We show how to fit ICONIX Process into an “ideal medium” agile project life cycle in this book’s companion volume, *Agile Development with ICONIX Process* (Apress, 2005).

as we gained experience in teaching it to clients, it became obvious that Jacobson's approach (use cases, robustness diagrams, and sequence diagrams) really worked pretty well.

In fact *it continually amazed us how well it worked on a wider and wider range of projects*. Experience in teaching the process convinced us that the “missing link” between requirements and design was the robustness diagram, and when UML was created and this diagram got relegated to an obscure appendix in the UML specification, we were seriously concerned that it would become a lost art form.

Our training business was given a bit of a boost when UML came into existence, as suddenly a lot more people were interested in how to do OOAD using a combined Jacobson/Rumbaugh/Booch approach, while our tools business (being Macintosh-based) didn't fare as well.

So ICONIX became a training company instead of a tools company, and, as our experience delivering training grew, there eventually came an opportunity to write a book: *Use Case Driven Object Modeling* (UCDOM), which I wrote with Kendall Scott. One of the reviewers of that book, Greg Wilson of *Dr. Dobbs Journal*, suggested that we write an example-intensive companion workbook, which we did. *Applying Use Case Driven Object Modeling* (AUCDOM), built around the Internet Bookstore example, was published a few years later.

## The Truth About Disambiguation

Meanwhile, we continued to deliver training, year after year, and (as far as we could tell) our clients continued to succeed with it. At least, they kept hiring us back to teach additional classes, which was the best metric we could think of for judging this.

OO technologies such as CORBA and COM appeared on the scene, followed by Java, DCOM, EJBs, C#, and .NET, and our use case-driven approach just kept right on working without skipping a beat. Occasionally we'd sit back and ponder why it hadn't broken, and it seemed like we (following in Ivar Jacobson's footsteps) had hit on a systematic approach that provided the answers to some fundamentally important questions that addressed the issue of how to get from use cases to code. This approach involved things like understanding all the scenarios and user interactions (both sunny- and rainy-day scenarios) before trying to do design; taking a little bit of extra time to disambiguate the behavior requirements before attacking detailed design issues; and focusing on “object discovery” first and “behavior allocation” (assigning operations to classes) later.

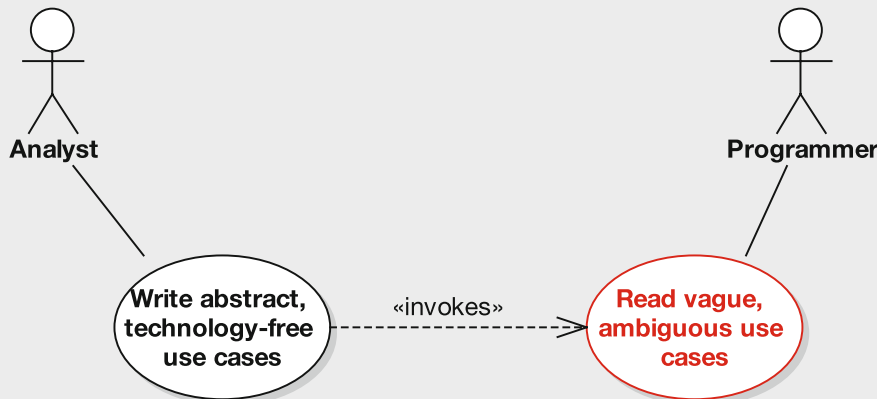
As the years went by and the number of training classes grew from dozens to hundreds, it became increasingly obvious that the notion of *disambiguating behavior requirements using robustness diagrams* was one of the most important “fundamental truths” that had emerged from Jacobson's work.

We can state that fundamental truth as follows: *one of the main reasons that programmers get frustrated by attempts to bring analysis and design (and especially use cases) into their projects is that they are generally given vague and ambiguous requirements to design against*. And the reason for so much ambiguity in use cases is that so many of the books and gurus out there preach “abstract, essential, technology-free, and implementation-independent” as the right way to write use cases.

To carry it one small step further, I'll make the following claim: if you hand a programmer an abstract, technology-free, implementation-independent, “essential” use case, that programmer will find the use case to be vague, ambiguous, incomplete, and therefore incorrect.

## FOOTLOOSE AND TECHNOLOGY-FREE

Without disambiguation, analysts write “essential, abstract, technology-free, and implementation-independent” use cases. The programmers who must read these use cases are, from their perspective, reading “vague, ambiguous, incomplete, and incorrect” use cases.



These use cases don't have enough detail to allow programmers to get to code while driving the OO design from the use cases. So, the use case-driven process doesn't work very well without robustness analysis (a technique we describe in detail in this book).

ICONIX Process seems to resonate better with programmers than many other approaches to use cases and UML/OOAD because it actually **forces the use cases into concrete, tangible, and specific statements of required system behavior** that programmers can deal with efficiently. If there's a secret to all of this, that's it.

## What's New

I took a writing detour for a few years (while continuing to deliver training in ICONIX Process) and Matt Stephens and I wrote *Extreme Programming Refactored: The Case Against XP<sup>2</sup>* and *Agile Modeling with ICONIX Process<sup>3</sup>* for Apress. Matt and I discovered that we work pretty well together, so he's joined me for the current effort. Meanwhile, *Use Case Driven Object Modeling* continues to sell and reached somewhere around 45,000 copies, including Chinese, Japanese, and Korean editions the last time I checked.

When we decided to do an update, we determined that there were a number of things that we could do that might justify a new edition (aka this book), including the following:

---

2. See [www.softwarereality.com/ExtremeProgrammingRefactored.jsp](http://www.softwarereality.com/ExtremeProgrammingRefactored.jsp).  
 3. See [www.softwarereality.com/AgileDevelopment.jsp](http://www.softwarereality.com/AgileDevelopment.jsp).

- Merge UCDDOM and AUDDOM into a single title, all based around the Internet Bookstore example
- Add student exercises, with the idea that some universities might start using the book as a text
- Create “top 10 to-do” lists, in addition to the “top 10 error” lists we already had
- Carry the Internet Bookstore forward all the way through code and test
- Update the process with a few new tricks we’ve learned over the years, and fully leverage some advances in modeling tools
- Update the book to be current with the new UML 2.0 specification (and with Ivar Jacobson’s new ideas on aspect-oriented programming [AOP])

As you’ll see, these goals have resulted in a typical chapter structure that’s in three parts: “Theory” (the process explained, using the Internet Bookstore as a running example), “Practice” (workbook exercises), and “More Practice” (student exercises). Matt went ahead and implemented a small Internet bookstore in Java, complete with unit tests driven from the use cases, which has allowed us to extend the book both in breadth and depth over the original titles (thanks, Matt).

We think that we’ve improved upon the original books in a number of ways, and we hope that you agree and like the result.

Doug Rosenberg  
ICONIX, [www.iconixsw.com](http://www.iconixsw.com)

# Introduction

*The difference between “theory” and “practice” is that in theory there is no difference between theory and practice, but in practice, there is.*

**D**oug has been using this phrase to open each and every training class for so long now that he’s forgotten where he first heard it. Matt did some research and found that it’s commonly credited to a Jan L. A. van de Snepscheut, who, in addition to having a wonderful name, was quite a distinguished professor at Caltech.<sup>4</sup>

Matt also found the quote attributed to Yogi Berra, who said, “In theory there is no difference between theory and practice. In practice there is.”<sup>5</sup> This makes us wonder if Professor Snepscheut might have been a baseball fan, or if Yogi made a practice of attending lectures at Caltech in the off-season, but no matter—they were both right.

Regardless of who said it first, we like to apply this statement to UML modeling, because, to be blunt, **UML is way too big**. A project trying to ingest all of UML into its working practices resembles a python that has just swallowed a pig. It’s going to take an awfully long time to digest, and your project probably can’t afford it.

*The Unified Modeling Language User Guide* by Grady Booch, James Rumbaugh, and Ivar Jacobson (Addison-Wesley, 1998) tells us that “**you can do 80% of your modeling with 20% of the UML**” somewhere after page 400.<sup>6</sup> They would have saved the industry many millions (billions?) of dollars and horrific cases of analysis paralysis (see the upcoming sidebar titled “The Mysterious Case of Analysis Paralysis”) if they had said that in the Introduction, but they didn’t. To compound the felony, **they never tell us which 20% of UML is the useful part**.

Most people that we meet *usually* want to apply UML in the up-front stages of their project. And most of them *usually* want to start their analysis process with use cases. So, in our search for the “**minimal, yet sufficient**” core subset of UML, we focus on the question, **How do you get from use cases to code?**

So, in *theory*, everything in the UML is useful, but in *practice*, a whole lot of people and projects need to know how to drive an OO software design from use cases. And they also need to know which diagrams from the UML directly help to accomplish this.

This book explains the minimalist, core subset of UML and the thought process behind using it to drive OO software designs from use cases (collectively referred to as ICONIX Process), with an eye toward the practical as opposed to the theoretical. We hope the journey will be both informative and entertaining.

---

4. Read Edgser W. Dijkstra’s “In Memoriam” for Professor Snepscheut at [www.cs.utexas.edu/users/EWD/transcriptions/EWD11xx/EWD1177.html](http://www.cs.utexas.edu/users/EWD/transcriptions/EWD11xx/EWD1177.html).

5. More Yogi-isms can be found here: [http://en.wikiquote.org/wiki/Yogi\\_Berra](http://en.wikiquote.org/wiki/Yogi_Berra). Yogi also said, “It’s tough to make predictions, especially about the future,” which clearly applies to software cost estimation, and “It was hard to have a conversation with anyone; there were so many people talking,” which is applicable to the “all the programmers in one big room” XP working environment.

6. See page 431 of the first edition.

## THE MYSTERIOUS CASE OF ANALYSIS PARALYSIS

It was a blustery, cold, rainy night at our flat on Baker Street. The howl of the wind whipping raindrops against the windows could be heard over Holmes' violin as I read the paper in front of the fireplace. Mrs. Hudson had just cleared away the dishes from our late supper of pork pie and beans, when Holmes suddenly paused in the aria he was playing, sat bolt upright in his chair, and exclaimed, "Watson, the game's afoot!"

A few moments later, our good friend Inspector Lestrade from Scotland Yard clattered up the stairs and burst in the doorway, exclaiming, "Thank goodness you're home, Mr. Holmes—you've got to come quickly!"

"Come in, Lestrade. Pray take a seat by the fire and tell us every detail," said Holmes.

"They're all dead, Mr. Holmes, every one of them—the whole project's dead! And no signs of violence, not a mark on any of them!" said Lestrade.

"Who's dead?" I asked.

"The entire staff of Scotland Yard's new automated fingerprint recognition system," Lestrade responded. "The whole technical staff . . . sitting dead right in the conference room . . . as if they'd been frozen to their chairs!"

"Has anything been touched?" asked Holmes.

"No, I've left the strictest instructions that the conference room be completely sealed off until you could inspect it," said Lestrade.

"Most excellent," murmured Holmes. "You are learning, Lestrade. Come along, Watson." Grabbing our coats and hats, we hastened down to Lestrade's waiting hansom cab.

We arrived shortly at Scotland Yard and were escorted to the conference room, where we were confronted by a bizarre and grisly death scene. Still in their seats, but struck down by some mysterious assassin, was the entire staff of the new automated fingerprint recognition project. Holmes walked around the room excitedly, his highly trained senses alert for any sort of clue. He paused at the whiteboard, and again at a stack of UML books on the table.

"You see, Mr. Holmes, they're all dead, and not a mark on any of them. How could a whole project just die like that?" asked Lestrade.

"Elementary, my dear Lestrade. A clear case of that obscure poison from the Amazon jungle known as *analysisparalysisflagrantis*. Perhaps you've seen my monograph on the topic? No? Tut, tut," murmured Holmes.

"But Holmes, how can you be sure?" I queried. "All I can see is these UML books scattered around the table. Here's one called ***Fully Dressed Use Cases: The Hallway Closet Approach*** by Professor Moriarty. It suggests you should stuff everything you can think of into your use cases, just like you do with the hallway closet," I said.

"You see, Watson, but you do not *observe*. Notice the three-day growth of beard on all the corpses, and the scrawls of <<includes>> and <<extends>> on the whiteboards?" asked Holmes.

"Sure enough, Mr. Holmes," said Lestrade. "Even the women have grown beards!"

"Great Scott!" I exclaimed. "Gives me the shivers."

"Analysis paralysis, Watson," said Holmes. "The second fastest killer of software projects, after *DoingTheSimplestThingThatCanPossiblyWork*, and nearly as dangerous. It's caused by a lethal overdose of formality and strict adherence to the UML semantics documentation. Moriarty's been up to his evil tricks again. You see the hollow expressions on the victims' faces, caused by interminable meetings spent debating topics of marginal uselessness. The despair and the anguish. The feverish search for a practical approach instead of highbrow theories. And all so easily avoidable," he sighed. "Come along, Watson, we have arrived too late."

We headed homeward toward Baker Street and the fireplace.

## Theory, in Practice

Each chapter in this book starts with the theory, and then explores said theory using the Internet Bookstore project. Over the course of the book, we'll demonstrate, in practice, the theory of getting from use cases to source code, using the Internet Bookstore as the main example throughout.

The practice doesn't stop there, though. This book also contains practical exercises of various types, which we describe here.

## Workbook Exercises

It's been clear for some time that the process of reviewing models is critically important and not well understood by many folks. So, in this book, we dissect the design of the Internet Bookstore, step by step, in great detail. This involves showing many common mistakes, and then showing the relevant pieces of the model with their mistakes corrected.

We've been teaching workshops using the Internet Bookstore example for many years, and as a result we have a rich source of classroom UML models with real student mistakes in them. We've collected some of our favorite mistakes—that is, the kinds of mistakes we saw repeated over and over again—and turned these into workbook exercises that you can find at the end of many of the chapters.

The following aspects are common to each set of exercises:

- There's an example diagram, with some errors intentionally left in.
- There's **a corrected version of the diagram a few pages later**. Corrections to the errors presented on the associated “bad” page are explicitly indicated; **explanations of the mistakes are provided in detail**.

## Student Exercises

At the end of most chapters in the “More Practice” section, you'll find student exercises to help you to test whether you truly “got” what the chapter is about. These exercises are in the form of more traditional numbered questions, and can thus be assigned as tasks for students.

For this reason, we don't provide the answers to these exercises in the book, although of course it's possible to *learn* the answers by reading and understanding the chapters! We do plan to provide some sort of “teacher's guide” material on the book's website, [www.iconixprocess.com](http://www.iconixprocess.com). The exact form of this teacher's guide has yet to be determined, so check the website for details.

## Inline Exercises Within the Chapters

Getting things right first time is great, but getting something wrong initially and then **learning from your mistakes** is a *much* better way to learn. Because of the way your brain is wired, you end up with a deeper understanding of the subject that way.

As we develop the example Internet Bookstore application through the course of the book, we don't just show the right thing to do next. We slip some “deliberate” mistakes into the diagrams, and then discover and correct them later (usually in the review chapters). However, unlike the “More Practice” exercises (where we do reveal the errors, in great detail), we don't

tell you precisely what the mistake is for these inline exercises. Instead, we provide a clue as to the nature of the error, and then invite you to scrutinize the diagram (and the relevant review chapter) and figure out what's wrong with it.

Trying to figure out what's wrong in the diagram is a good way to learn, but there's another element to this. Notice that we didn't say "the answer can be found on page 141" or "check the list of answers at the end of this chapter," as that would be too easy. An important part of the learning process is in searching through the next chapter, looking for the paragraph that reveals the answer. You'll be surprised how well you learn while you're hunting down the solution to a particular problem.

Cool set of premises, aren't they? We're not aware of another book like this one, and we're hoping you'll find it useful in your efforts to apply use case–driven object modeling with UML.

## Top 10 Things People Who Use ICONIX Process Like About It

Each chapter in this book kicks off with a top 10 list of guidelines, and the first half of each chapter is structured around its top 10 list. For this Introduction, we've put together a list of the top 10 comments that we've heard from clients who've applied ICONIX Process on their own projects.

**10.** *The process uses a core subset of UML.*

(We'd rather learn 4 diagrams than 14 . . .)

**9.** *It actually gets me all the way to code.*

(I have 13 use case books on my shelf that don't get within 50 miles of code.)

**8.** *It's traceable from one step to the next.*

**7.** *It addresses both sunny- and rainy-day scenarios.*

(If another one of my programmers tells me they're "Doing The Simplest Thing That Could Possibly Work" [DTSTTCPW], I think I'm gonna scream.)

**6.** *It assumes that the requirements I'm initially given are vague, ambiguous, incomplete, and incorrect.*

(Have Doug and Matt actually met our business analysts?)

**5.** *It actually drives the OO design from the use cases.*

(I know RUP says that it's use case–driven, but I get lost somewhere around the Elaboration phase.)

**4.** *It works well in an "agile" (short iteration, small increment) environment.*

(I wish somebody would write a book on how to do Agile/ICONIX, though.)<sup>7</sup>

---

7. We did: *Agile Development with ICONIX Process* (Apress, 2005).



3. *It doesn't drown me in five-syllable buzzwords.*

(What about multiple polymorphic inheritance, anyway?)<sup>8</sup>

2. *It operates at a tangible level where the use cases talk about what the users are doing on the screens, and there are no huge use case templates.*

(In other words, the use cases aren't abstract, essential, technology-free, or implementation independent.)

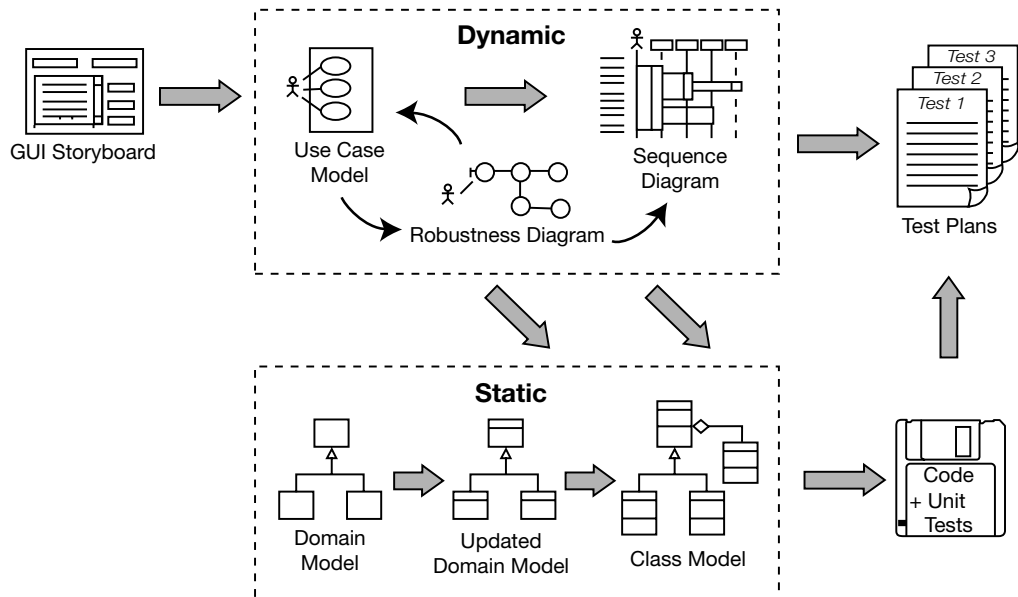
1. *It's a practical approach that's been proven to work in the real world, on hundreds of projects.*

---

8. Although we do have fun making up new buzzwords and phrases, like “disintermangling dysfunctional requirements.”



# Introduction to ICONIX Process



*One process is much larger  
 And the other's way too small  
 And the full UML that OMG gives you  
 Is incomprehensible to all . . .*

(Sing to the tune of "Go Ask Alice" by Jefferson Airplane)

In theory, every single aspect of the UML is potentially useful, but in practice, there never seems to be enough time to do modeling, analysis, and design. There's always pressure from management to jump to code, to start coding prematurely because progress on software projects tends to get measured by how much code exists. ICONIX Process, as shown in the chapter's opening figure, is a minimalist, streamlined approach that focuses on that area that lies in between use cases and code. Its emphasis is on what needs to happen at that point in the life cycle where you're starting out: you have a start on some use cases, and now you need to do good analysis and design.

## WHEN TO USE A COOKBOOK

There's a growing misconception in software development that cookbook approaches to software development don't work. We agree with this to an extent, because analysis and programming are massive, highly complex fields, and the number of different software project types is roughly equal to the number of software projects. However, we firmly believe that analysis and design can—and in fact should—be a specific sequence of repeatable steps. These steps aren't set in stone (i.e., they can be tailored), but it helps to have them there. In a world filled with doubt and uncertainty, it's nice to have a clearly defined sequence of “how-to” steps to refer back to.

Way back in the pre-UML days when Doug first started teaching a unified Booch/Rumbaugh/Jacobson modeling approach (around 1992/1993), one of his early training clients encouraged him to “write a cookbook, because my people like following cookbook approaches.” While many have claimed that it's impossible to codify object-oriented analysis and design (OOAD) practices into a simple, repeatable set of steps (and it probably isn't possible in its entirety), ICONIX Process probably comes as close as anything out there to a cookbook approach to OOAD.

While there's still room for significant flexibility within the approach (e.g., adding in state or activity diagrams), ICONIX Process lays down a simple, minimal set of steps that generally lead to pretty good results. These results have proven to be consistent and repeatable over the last 12 years.

## ICONIX Process in Theory

In this section we provide an overview of ICONIX Process, showing how all the activities fit together. We'll start with a very high-level view—kind of an overview of the overview—and then we'll examine each activity in more detail. As you're walking through the overview, keep referring back to the process diagram at the start of this chapter, to see how each part fits into the overall process.

### Overview: Getting from Use Cases to Source Code

The diagram at the start of this chapter gives an overview of ICONIX Process. (We'll repeat this diagram at the start of each chapter, with the relevant section of the diagram shown in red.) As you can see from the diagram, ICONIX Process is divided into *dynamic* and *static* workflows, which are highly iterative: you might go through one iteration of the whole process for a small batch of use cases (perhaps a couple of packages' worth, which isn't a huge amount given that each use case is only a couple of paragraphs), all the way to source code and unit tests. For this reason, ICONIX Process is well suited to agile projects, where swift feedback is needed on such factors as the requirements, the design, and estimates.

Let's walk through the steps that we'll cover in the course of this book. The **items in red** correspond with the subtitles in this section (pretty slick, huh?).

As with any project, at some stage early on you begin exploring and defining the requirements. Note that within each phase there's a degree of parallelism, so all the activities in the requirements definition phase go on sort of overlapped and interleaved until they're ready.

---

■ **Note** There are many different types of requirements (e.g., nonfunctional requirements such as scalability). However, at a process level, we distinguish between *functional requirements* and *behavioral requirements*.

---

## 1. REQUIREMENTS

- a. **Functional requirements:** Define what the system should be capable of doing. Depending on how your project is organized, either you'll be involved in creating the functional requirements or the requirements will be "handed down from on high" by a customer or a team of business analysts.
- b. **Domain modeling:** Understand the problem space in unambiguous terms.
- c. **Behavioral requirements:** Define how the user and the system will interact (i.e., write the first-draft use cases). We recommend that you start with a GUI prototype (**storyboarding the GUI**) and identify all the use cases you're going to implement, or at least come up with a first-pass list of use cases, which you would reasonably expect to change as you explore the requirements in more depth.
- d. **Milestone 1: Requirements Review:** Make sure that the use case text matches the customer's expectations. Note that you might review the use cases in small batches, just prior to designing them.

Then in each iteration (i.e., for a small batch of use cases), you do the following.

## 2. ANALYSIS/PRELIMINARY DESIGN

- a. **Robustness analysis:** Draw a robustness diagram (an "object picture" of the steps in a use case), rewriting the use case text as you go.
- b. Update the domain model while you're writing the use case and drawing the robustness diagram. Here you will discover missing classes, correct ambiguities, and add attributes to the domain objects (e.g., identify that a Book object has a Title, Author, Synopsis, etc.).
- c. Name all the logical software functions (**controllers**) needed to make the use case work.
- d. Rewrite the first draft use cases.

## 3. Milestone 2: Preliminary Design Review (PDR)

## 4. DETAILED DESIGN

- a. **Sequence diagramming:** Draw a sequence diagram (one sequence diagram per use case) to show *in detail* how you're going to implement the use case. The primary function of sequence diagramming is to allocate behavior to your classes.

- b. Update the domain model while you're drawing the sequence diagram, and add operations<sup>1</sup> to the domain objects. By this stage, the domain objects are really domain classes, or *entities*, and the domain model should be fast becoming a *static model*, or *class diagram*—a crucial part of your detailed design.
  - c. **Clean up the static model.**
- 5. **Milestone 3: Critical Design Review (CDR)**
- 6. **IMPLEMENTATION**
  - a. **Coding/unit testing:** Write the code and the unit tests. (Or, depending on your preferences, write the unit tests and then the code.<sup>2</sup>)
  - b. **Integration and scenario testing:** Base the integration tests on the use cases, so that you're testing both the basic course and the alternate courses.
  - c. Perform a **Code Review and Model Update** to prepare for the next round of development work.

For most of the rest of this chapter, we describe these steps in a little more detail. Throughout the rest of the book, we describe these steps in *much* greater detail, and provide lots of examples and exercises to help you understand how best to apply them to your own project.

## Requirements

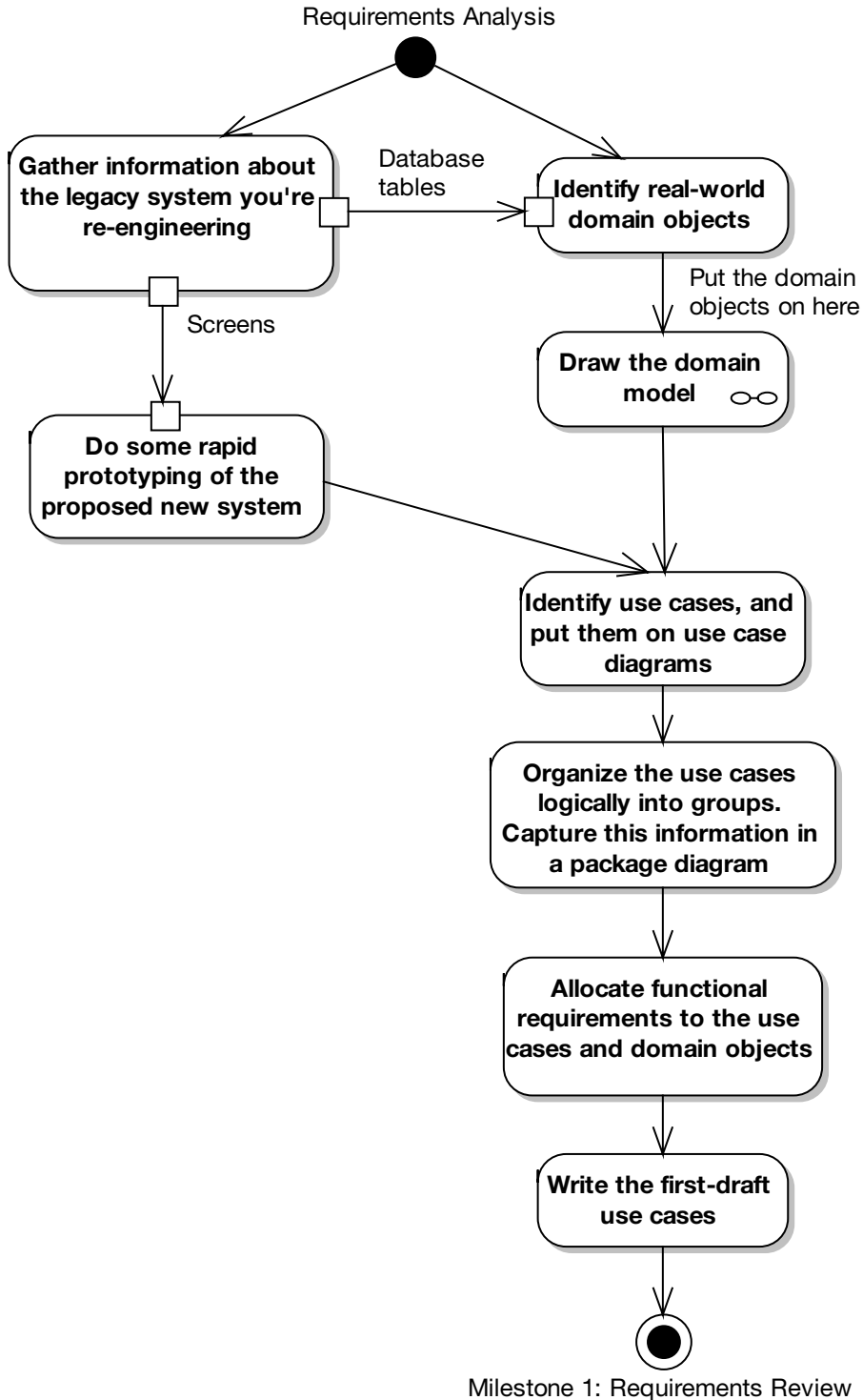
Figure 1-1 shows the steps involved in defining the **behavioral requirements**—that is, drawing the initial domain model and writing the first-draft use cases.

The steps shown in Figure 1-1 are covered in Chapters 2, 3, and 4.

---

1. Also called methods, functions, or messages, depending which programming language you use.

2. For Test-Driven Development (TDD) fans, in Chapter 12 we illustrate a method of incorporating the “test first” approach into ICONIX Process. The result is essentially “Design-Driven Testing.”



**Figure 1-1.** Requirements analysis