

The Crystal Series for Software Developers  
Alistair Cockburn, Series Editor

# Writing Effective Use Cases



**Alistair Cockburn**

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

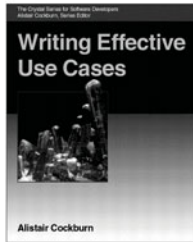


## **The Writing Process**

1. Name the system scope and boundaries.  
*Track changes to this initial context diagram with the in/out list.*
2. Brainstorm and list the primary actors.  
*Find every human and non-human primary actor, over the life of the system.*
3. Brainstorm and exhaustively list user goals for the system.  
*The initial Actor-Goal List is now available.*
4. Capture the outermost summary use cases to see who really cares.  
*Check for an outermost use case for each primary actor.*
5. Reconsider and revise the summary use cases. Add, subtract, or merge goals.  
*Double-check for time-based triggers and other events at the system boundary.*
6. Select one use case to expand.  
*Consider writing a narrative to learn the material.*
7. Capture stakeholders and interests, preconditions and guarantees.  
*The system will ensure the preconditions and guarantee the interests.*
8. Write the main success scenario (MSS).  
*Use 3 to 9 steps to meet all interests and guarantees.*
9. Brainstorm and exhaustively list the extension conditions.  
*Include all that the system can detect and must handle.*
10. Write the extension-handling steps.  
*Each will end back in the MSS, at a separate success exit, or in failure.*
11. Extract complex flows to sub use cases; merge trivial sub use cases.  
*Extracting a sub use case is easy, but it adds cost to the project.*
12. Readjust the set: add, subtract, merge, as needed.  
*Check for readability, completeness, and meeting stakeholders' interests.*

# The Agile Software Development Series

Alistair Cockburn and Jim Highsmith, Series Editors



◆◆ Addison-Wesley

Visit [informit.com/agileseries](http://informit.com/agileseries) for a complete list of available publications.

**A**gile software development centers on four values, which are identified in the Agile Alliance's Manifesto\*:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

The development of Agile software requires innovation and responsiveness, based on generating and sharing knowledge within a development team and with the customer. Agile software developers draw on the strengths of customers, users, and developers to find just enough process to balance quality and agility.

The books in The Agile Software Development Series focus on sharing the experiences of such Agile developers. Individual books address individual techniques (such as Use Cases), group techniques (such as collaborative decision making), and proven solutions to different problems from a variety of organizational cultures. The result is a core of Agile best practices that will enrich your experiences and improve your work.

\* © 2001, Authors of the Agile Manifesto

◆◆ Addison-Wesley

[informIT.com](http://informit.com)

Safari<sup>®</sup>  
Books Online

---

*Writing Effective  
Use Cases*

*This page intentionally left blank*

---

# *Writing Effective Use Cases*

***Alistair Cockburn***

. σ  
ττ

**Addison-Wesley**

Boston • San Francisco • New York • Toronto • Montreal  
London • Munich • Paris • Madrid  
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and we were aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Copyright © 2001 by Addison-Wesley.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher. Printed in the United States of America. Published simultaneously in Canada.

The publisher offers discounts on this book when ordered in quantity for special sales. For more information, please contact:

Pearson Education Corporate Sales Division  
One Lake Street  
Upper Saddle River, NJ 07458  
(800) 382-3419  
*corpsales@pearsontechgroup.com*

Visit us on the Web at *www.awl.com/cseng/*

#### Library of Congress Cataloging-in-Publication Data

Cockburn, Alistair.

Writing effective use cases / Alistair Cockburn.

p. cm. -- (The Crystal Collection for software professionals)

Includes bibliographical references and index.

ISBN 0-201-70225-8 (alk. paper)

1. Application software—Development. 2. Use cases (Systems engineering)

I. Title. II. Series.

QA76.76A65 C63 2000

005.3--dc2

00-040179

Text printed in the United States on recycled paper at Edwards Brothers in Ann Arbor, Michigan.

Twenty-third printing, April 2011

# Contents

---

Preface     *xix*  
Acknowledgments     *xxiii*

**Chapter 1** *Introduction* **1**

---

1.1 What Is a Use Case (More or Less)? . . . . . 1  
    Use Case 1   📦 *Buy Stocks over the Web* 🚀 . . . . . 4  
    Use Case 2   🏠 *Get Paid for Car Accident* 📄 . . . . . 5  
    Use Case 3   📦 *Register Arrival of a Box* 🚀 . . . . . 6

1.2 Your Use Case Is Not My Use Case . . . . . 7  
    Use Case 4   🏠 *Buy Something (Casual Version)* 📄 . . . . . 9  
    Use Case 5   🏠 *Buy Something (Fully Dressed Version)* 📄 . . . . . 9

◆ **Steve Adolph: “Discovering” Requirements in New Territory** . . . . . 12

1.3 Requirements and Use Cases . . . . . 13  
    Use Cases as Project-Linking Structure . . . . . 14  
        Figure 1.1 The “Hub-and-Spoke” model of requirements. . . . . 15

1.4 When Use Cases Add Value . . . . . 15

1.5 Manage Your Energy . . . . . 16

1.6 Warm Up with a Usage Narrative . . . . . 17

◆ **Usage Narrative: Getting “Fast Cash”** . . . . . 18













1.7 Exercises. . . . . 19

















**Part 1 The Use Case Body Parts** 21**Chapter 2 The Use Case as a Contract for Behavior** 23

2.1 Interactions between Actors with Goals . . . . .	23
Actors Have Goals . . . . .	23
Figure 2.1 An actor with a goal calls on the responsibilities of another. . . . .	24
Goals Can Fail . . . . .	25
Interactions Are Compound. . . . .	25
A Use Case Collects Scenarios . . . . .	27
Figure 2.2 Striped trousers: Scenarios succeed or fail. . . . .	28
Figure 2.3 The striped trousers showing subgoals. . . . .	29
2.2 Contract between Stakeholders with Interests. . . . .	29
Figure 2.4 The SuD serves the primary actor, protecting offstage stakeholders . . . .	30
2.3 The Graphical Model. . . . .	31
Figure 2.5 Actors and stakeholders. . . . .	32
Figure 2.6 Behavior. . . . .	32
Figure 2.7 Use Case as responsibility invocation. . . . .	33
Figure 2.8 Interactions as composite. . . . .	33



**Chapter 3 Scope** 35



Table 3.1 A Sample In/Out List . . . . .	36
3.1 Functional Scope . . . . .	36
The Actor-Goal List. . . . .	36
Table 3.2 A Sample Actor-Goal List. . . . .	37
The Use Case Briefs. . . . .	37
Table 3.3 Sample Use Case Briefs. . . . .	38
3.2 Design Scope . . . . .	38
Figure 3.1 Design scope can be any size. . . . .	40
Using Graphical Icons to Highlight the Design Scope . . . . .	40
Design Scope Examples . . . . .	41
Enterprise-to-System Examples . . . . .	41
Use Case 6  Add New Service (Enterprise)  . . . . .	42
Use Case 7  Add New Service (Acura)  . . . . .	42
Many Computers to One Application. . . . .	43
Use Case 8  Enter and Update Requests (Joint System)  . . . . .	43
Use Case 9  Add New Service (into Acura)  . . . . .	44
Use Case 10  Note New Service Request (in BSSO)  . . . . .	44
Use Case 11  Update Service Request (in BSSO)  . . . . .	44



Use Case 12	 <i>Note Updated Request (in Acura)</i> 	44
Figure 3.2	Use case diagrams for Acura–BSSO..	45
Figure 3.3	A combined use case diagram for Acura-BSSO..	45
Nuts and Bolts Use Cases .....		
Use Case 13	 <i>Serialize Access to a Resource</i> 	46
Use Case 14	 <i>Apply a Lock Conversion Policy</i> 	47
Use Case 15	 <i>Apply an Access Compatibility Policy</i> 	48
Use Case 16	 <i>Apply an Access Selection Policy</i> 	48
Use Case 17	 <i>Make Service Client Wait for Resource Access</i> 	49
3.3	The Outermost Use Cases .....	49
3.4	Using the Scope-Defining Work Products .....	51
3.5	Exercises .....	51
<hr/>		
<b>Chapter 4</b>	<b><i>Stakeholders and Actors</i></b>	<b>53</b>
<hr/>		
4.1	Stakeholders .....	53
4.2	The Primary Actor .....	54
	Why Primary Actors Are Unimportant (and Important) .....	55
	Actors versus Roles .....	57
	Characterizing the Primary Actors .....	58
	Table 4.1 A Sample Actor Profile Table .....	56
4.3	Supporting Actors .....	59
4.4	The System Under Discussion .....	59
4.5	Internal Actors and White-Box Use Cases .....	59
4.6	Exercises .....	60
<hr/>		
<b>Chapter 5</b>	<b><i>Three Named Goal Levels</i></b>	<b>61</b>
<hr/>		
	Figure 5.1 Use case levels .....	62
5.1	User Goals (Blue, Sea-Level) .....	62
	Two Levels of Blue .....	63
5.2	Summary Level (White, Cloud/ Kite) .....	64
	Use Case 18  <i>Operate an Insurance Policy+</i> 	65
	The Outermost Use Cases Revisited .....	65
5.3	Subfunctions (Indigo/Black, Underwater/Clam) .....	66
	Summarizing Goal Levels .....	66
5.4	Using Graphical Icons to Highlight Goal Levels .....	67
5.5	Finding the Right Goal Level .....	68
	Finding the User's Goal .....	68



*Raising and Lowering Goal Levels* . . . . . 69  
*Figure 5.2 Ask “why” to shift levels* . . . . . 69



5.6 A Longer Writing Sample: “Handle a Claim” at Several Levels . . . . . 70

    Use Case 19  *Handle a Claim (Business)*  . . . . . 71

    Use Case 20  *Evaluate Work Comp Claim*  . . . . . 72

    Use Case 21  *Handle a Claim (Systems)* +  . . . . . 73

    Use Case 22  *Register a Loss*  . . . . . 75

    Use Case 23  *Find a Whatever (Problem Statement)*  . . . . . 79

5.7 Exercises . . . . . 79

**Chapter 6** *Preconditions, Triggers, and Guarantees* **81**

---

6.1 Preconditions . . . . . 81

6.2 Minimal Guarantees . . . . . 83

6.3 Success Guarantee . . . . . 84

6.4 Triggers . . . . . 84

6.5 Exercises . . . . . 85

**Chapter 7** *Scenarios and Steps* **87**

---

7.1 The Main Success Scenario . . . . . 87

    The Common Surrounding Structure . . . . . 87

    The Scenario Body . . . . . 89

7.2 Action Steps . . . . . 90

    Guidelines . . . . . 90

    Guideline 1: Use Simple Grammar . . . . . 90

    Guideline 2: Show Clearly “Who Has the Ball” . . . . . 90

    Guideline 3: Write from a Bird’s Eye View . . . . . 91

    Guideline 4: Show the Process Moving Forward . . . . . 91

    Guideline 5: Show the Actor’s Intent, Not the Movements . . . . . 92

    Guideline 6: Include a “Reasonable” Set of Actions . . . . . 93

        Figure 7.1 A transaction has four parts . . . . . 93

    Guideline 7: “Validate,” Don’t “Check Whether” . . . . . 95





    Guideline 8: Optionally Mention the Timing . . . . . 95











    Guideline 9: Idiom: “User Has System A Kick System B” . . . . . 96





    Guideline 10: Idiom: “Do Steps x–y until Condition” . . . . . 96

    To Number or Not to Number . . . . . 97

7.3 Exercises . . . . . 98

<b>Chapter 8</b>	<i>Extensions</i>	<b>99</b>
8.1	Extension Basics . . . . .	99
8.2	The Extension Conditions . . . . .	100
	Brainstorm All Conceivable Failures and Alternative Courses . . . . .	101
	Guideline 11: Make the Condition Say What Was Detected . . . . .	102
	Rationalize the Extensions List . . . . .	104
	Rollup Failures . . . . .	105
8.3	Extension Handling . . . . .	106
	Guideline 12: Indent Condition Handling . . . . .	108
	Failures within Failures . . . . .	109
	Creating a New Use Case from an Extension . . . . .	109
8.4	Exercises . . . . .	110
<b>Chapter 9</b>	<i>Technology and Data Variations</i>	<b>111</b>
	Figure 9.1 Technology variations using specialization in UML . . . . .	112
<b>Chapter 10</b>	<i>Linking Use Cases</i>	<b>113</b>
10.1	Sub Use Cases . . . . .	113
10.2	Extension Use Cases . . . . .	114
	Figure 10.1 UML diagram of extension use cases . . . . .	115
	When to Use Extension Use Cases . . . . .	116
10.3	Exercises . . . . .	117
<b>Chapter 11</b>	<i>Use Case Formats</i>	<b>119</b>
11.1	Formats to Choose From . . . . .	119
	Fully Dressed . . . . .	119
	Use Case 24 <i>Fully Dressed Use Case Template</i> <name> . . . . .	119
	Casual . . . . .	120
	Use Case 25  <i>Actually Login (Casual Version)</i>  . . . . .	120
	One-Column Table . . . . .	121
	Table 11.1 One-Column Table Format of a Use Case . . . . .	121
	Two-Column Table . . . . .	122
	Table 11.2 Two-Column Table . . . . .	122
	RUP Style . . . . .	123
	Use Case 26  <i>Register for Courses</i>  . . . . .	124

- If-Statement Style..... 126
- Occam Style..... 126
- Diagram Style ..... 127
- The UML Use Case Diagram..... 128
- 11.2 Forces Affecting Use Case Writing Styles..... 128
  - Consistency ..... 130
  - Complexity..... 130
- 11.3 Standards for Five Project Types..... 132
  - For Requirements Elicitation ..... 133
    - Use Case 27  *Elicitation Template—Oble a New Biscum*  ..... 133
  - For Business Process Modeling..... 134
    - Use Case 28  *Business Process Template—Symp a Carstromming*  ..... 134
  - For Sizing the Requirements..... 135
    - Use Case 29  *Sizing Template—Burble the Tramling*  ..... 135
  - For a Short, High-Pressure Project ..... 136
    - Use Case 30  *High-Pressure Template: Kree a Ranfath*  ..... 136
  - For Detailed Functional Requirements ..... 137
    - Use Case 31  *Use Case Name—Nathorize a Permion*  ..... 137
- 11.4 Conclusion ..... 137
- 11.5 Exercise ..... 138

<b>Part 2</b>	<i>Frequently Discussed Topics</i>	<b>139</b>
<b>Chapter 12</b>	<i>When Are We Done?</i>	<b>141</b>
	On Being Done .....	142
<b>Chapter 13</b>	<i>Scaling Up to Many Use Cases</i>	<b>143</b>
	Say Less about Each One (Low-Precision Representation) .....	143
	Create Clusters of Use Cases .....	143
<b>Chapter 14</b>	<i>CRUD and Parameterized Use Cases</i>	<b>145</b>
14.1	CRUD Use Cases .....	145
	Use Case 32  <i>Manage Reports</i>  .....	146
	Use Case 33  <i>Save Report</i>  .....	148
14.2	Parameterized Use Cases .....	150
<b>Chapter 15</b>	<i>Business Process Modeling</i>	<b>153</b>
15.1	Modeling versus Designing .....	153
	Work from the Core Business .....	154
	Figure 15.1 Core business black box .....	155
	Figure 15.2 New business design in white box .....	155
	Work from Business Process to Technology .....	155
	Figure 15.3 New business design in white box (again) .....	156
	Figure 15.4 New business process in black-box system use cases .....	156
	Work from Technology to Business Process .....	157
15.2	Linking Business and System Use Cases .....	157
	◆ <b>Rusty Walters: Business Modeling and System Requirements</b> .....	159
<b>Chapter 16</b>	<i>The Missing Requirements</i>	<b>161</b>
16.1	Precision in Data Requirements .....	162
16.2	Cross-linking from Use Cases to Other Requirements .....	164
	Figure 16.1 Recap of Figure 1.1, “Hub-and-Spoke” model of requirements .....	164

**Chapter 17** *Use Cases in the Overall Process* **167**

---

17.1 Use Cases in Project Organization. . . . . 167

    Organize by Use Case Titles . . . . . 167

        Table 17.1 Sample Planning Table. . . . . 168

    Handle Use Cases Crossing Releases. . . . . 169

    Deliver Complete Scenarios. . . . . 170

17.2 Use Cases to Task or Feature Lists. . . . . 171



    Use Case 34  *Capture Trade-In*  . . . . . 172

        Table 17.2 Work List for Capture Trade-In . . . . . 173

17.3 Use Cases to Design. . . . . 174

    A Special Note to Object-Oriented Designers . . . . . 176

17.4 Use Cases to UI Design . . . . . 177

17.5 Use Cases to Test Cases. . . . . 178



    Use Case 35  *Order Goods, Generate Invoice (Testing Example)*  . . . . . 178

        Table 17.3 Main Success Scenario Tests (Good Credit Risk) . . . . . 179

        Table 17.4 Main Success Scenario Tests (Bad Credit Risk). . . . . 180

17.6 The Actual Writing . . . . . 180

    A Branch-and-Join Process . . . . . 180

    Time Required per Use Case . . . . . 184

    Collecting Use Cases from Large Groups . . . . . 184

◆ **Andy Kraus: Collecting Use Cases from a Large, Diverse Lay Group.** . . . . . 184

**Chapter 18** *Use Case Briefs and Extreme Programming* **187**

---

**Chapter 19** *Mistakes Fixed* **189**

---

19.1 No System. . . . . 189



19.2 No Primary Actor. . . . . 190



19.3 Too Many User Interface Details. . . . . 191

19.4 Very Low Goal Levels . . . . . 192

19.5 Purpose and Content Not Aligned . . . . . 193

19.6 Advanced Example of Too Much UI . . . . . 194

    Use Case 36  *Research a Solution—Before*  . . . . . 194

    Use Case 37  *Research Possible Solutions—After*  . . . . . 199

<b>Part 3</b>	<b>Reminders for the Busy</b>	<b>203</b>
<b>Chapter 20</b>	<b>Reminders for Each Use Case</b>	<b>205</b>
Reminder 1:	A Use Case Is a Prose Essay . . . . .	205
Reminder 2:	Make the Use Case Easy to Read. . . . .	205
Reminder 3:	Just One Sentence Form . . . . .	206
Reminder 4:	“Include” Sub Use Cases . . . . .	207
Reminder 5:	Who Has the Ball? . . . . .	207
Reminder 6:	Get the Goal Level Right . . . . .	208
Figure 20.1	Ask “why” to shift levels. . . . .	208
Reminder 7:	Keep the GUI Out . . . . .	209
Reminder 8:	Two Endings . . . . .	209
Reminder 9:	Stakeholders Need Guarantees . . . . .	210
Reminder 10:	Preconditions . . . . .	211
Reminder 11:	Pass/Fail Tests for One Use Case . . . . .	211
Table 20.1	Pass/Fail Tests for One Use Case. . . . .	212
<b>Chapter 21</b>	<b>Reminders for the Use Case Set</b>	<b>215</b>
Reminder 12:	An Ever-Unfolding Story . . . . .	215
Reminder 13:	Both Corporate Scope and System Scope. . . . .	216
Reminder 14:	Core Values and Variations . . . . .	216
Reminder 15:	Quality Questions across the Use Case Set . . . . .	219
<b>Chapter 22</b>	<b>Reminders for Working on the Use Cases</b>	<b>221</b>
Reminder 16:	It’s Just Chapter 3 (Where’s Chapter 4?) . . . . .	221
Reminder 17:	Work Breadth First . . . . .	221
Figure 22.1	Work expands with precision. . . . .	222
Reminder 18:	The 12-Step Recipe . . . . .	223
Reminder 19:	Know the Cost of Mistakes. . . . .	223
Reminder 20:	Blue Jeans Preferred . . . . .	224
Reminder 21:	Handle Failures . . . . .	225
Reminder 22:	Job Titles Sooner <i>and</i> Later . . . . .	225
Reminder 23:	Actors Play Roles . . . . .	226
Reminder 24:	The Great Drawing Hoax . . . . .	227
Figure 22.2	“Mommy, I want to go home.”. . . . .	227
Figure 22.3	Context diagram in ellipse figure form. . . . .	228
Table 22.1	Actor-Goal List for Context Diagram. . . . .	228
Reminder 25:	The Great Tool Debate. . . . .	229
Reminder 26:	Project Planning Using Titles and Briefs . . . . .	230



## Appendices

<b>Appendix A</b>	<i>Use Cases in UML</i>	<b>233</b>
<hr/>		
A.1	Ellipses and Stick Figures . . . . .	233
A.2	UML's Includes Relation . . . . .	234
	Figure A.1 Drawing Includes. . . . .	234
	Guideline 13: Draw Higher Goals Higher. . . . .	235
A.3	UML's Extends Relation . . . . .	235
	Figure A.2 Drawing Extends. . . . .	236
	Guideline 14: Draw Extending Use Cases Lower . . . . .	236
	Guideline 15: Use Different Arrow Shapes . . . . .	236
	Correct Use of Extends . . . . .	237
	Figure A.3 Three interrupting use cases extending a base use case . . . . .	237
	Extension Points . . . . .	237
A.4	UML's Generalizes Relations . . . . .	239
	Correct Use of Generalizes. . . . .	239
	Figure A.4 Drawing Generalizes. . . . .	240
	Guideline 16: Draw General Goals Higher . . . . .	240
	Hazards of Generalizes . . . . .	240
	Figure A.5 Hazardous generalization—closing a big deal. . . . .	241
	Figure A.6 Correctly closing a big deal. . . . .	241
A.5	Subordinate versus Sub Use Cases . . . . .	242
A.6	Drawing Use Case Diagrams . . . . .	242
	Guideline 17: User Goals in a Context Diagram . . . . .	243
	Guideline 18: Supporting Actors on the Right . . . . .	243
A.7	Write Text-based Use Cases Instead. . . . .	243
<hr/>		
<b>Appendix B</b>	<i>Answers to (Some) Exercises</i>	<b>245</b>
	Chapter 3, page 51. . . . .	245
	Exercise 3.1	
	Exercise 3.2	
	Figure B.1 Design scopes for the ATM . . . . .	245
	Chapter 4, page 60. . . . .	246
	Exercise 4.2	
	Exercise 4.3	
	Chapter 5, page 79. . . . .	247
	Exercise 5.1	
	Exercise 5.2	

Chapter 6, page 85. . . . . 248  
 Exercise 6.1  
 Exercise 6.4  
 Chapter 7, page 98. . . . . 249  
 Exercise 7.1  
 Exercise 7.2  
 Exercise 7.4  
 Use Case 38  *Use the Order Processing System*  . . . . . 250  
 Chapter 8, page 110 . . . . . 252  
 Exercise 8.1  
 Exercise 8.5  
 Use Case 39  *Buy Stocks Over the Web*  . . . . . 251  
 Chapter 11, page 138 . . . . . 252  
 Exercise 11.1  
 Use Case 40  *Perform Clean Spark Plugs Service*  . . . . . 252

**Appendix C Glossary** **253**

---

Main Terms . . . . . 253  
 Use Case Types. . . . . 255  
 Diagrams . . . . . 256

**Appendix D Readings** **257**

---

Books Referenced in the Text . . . . . 257  
 Articles Referenced in the Text . . . . . 257  
 Useful Online Resources. . . . . 258

**Index** **259**

---

*This page intentionally left blank*

# Preface

---

More and more people are writing use cases, for behavioral requirements, for software systems or to describe business processes. It all seems easy enough—just write about using the system. But, faced with writing, one suddenly confronts the question, “Exactly what am I supposed to write—how much, how little, what details?” That turns out to be a difficult question to answer. The problem is that writing use cases is fundamentally an exercise in writing prose essays, with all the difficulties in articulating *good* that comes with prose writing in general. It is hard enough to say what a good use case looks like, but we really want to know something harder: how to write them so they will come out being good.

These pages contain the guidelines I use in my use case writing and in coaching: how a person might think, what he or she might observe, to end up with a better use case and use case set.

I include examples of good and bad use cases, plausible ways of writing differently, and, best of all, the good news that a use case need not be the *best* to be *useful*. Even mediocre use cases are useful, more so than are many of the competing requirements files being written. So relax, write something readable, and you will have done your organization a service.

## **Audience**

This book is predominantly aimed at industry professionals who read and study alone, and is therefore organized as a self-study guide. It contains introductory through advanced material: concepts, examples, reminders, and exercises (some with answers, some without).

Writing coaches should find suitable explanations and samples to show their teams. Course designers should be able to build course material around the book, issuing

reading assignments as needed. (However, as I include answers to many exercises, they will have to construct their own exam material. :-) )

## **Organization**

The book is organized as a general introduction to use cases followed by a close description of the use case body parts, frequently asked questions, reminders for the busy, and end notes.

The **Introduction** contains an initial presentation of key notions, to get the discussion rolling: “What does a use case look like?,” “When do I write one?,” and “What variations are legal?” The brief answer is that they look different depending on when, where, with whom, and why you are writing them. That discussion begins in this early chapter, and continues throughout the book

**Part 1, The Use Case Body Parts**, contains chapters for each of the major concepts that need to be mastered, and parts of the template that should be written. These include “The Use Case as a Contract for Behavior,” “Scope,” “Stakeholders and Actors,” “Three Named Goal Levels,” “Preconditions, Triggers, and Guarantees,” “Scenarios and Steps,” “Extensions,” “Technology and Data Variations,” “Linking Use Cases,” and “Use Case Formats.”

**Part 2, Frequently Discussed Topics**, addresses particular topics that come up repeatedly: “When Are We Done?,” “Scaling Up to Many Use Cases,” “CRUD and Parameterized Use Cases,” “Business Process Modeling,” “The Missing Requirements,” “Use Cases in the Overall Process,” “Use Case Briefs and eXtreme Programming,” and “Mistakes Fixed.”

**Part 3, Reminders for the Busy**, contains a set of reminders for those who have finished reading the book, or already know this material and want to refer back to key ideas. The chapters are organized as “Reminders for Each Use Case,” “Reminders for the Use Case Set,” and “Reminders for Working on the Use Cases.”

There are four appendices: Appendix A discusses “Use Cases in UML” and Appendix B contains “Answers to (Some) Exercises.” The book concludes with Appendix C, Glossary; and a list of materials used while writing, Appendix D, Readings.

## **Heritage of the Ideas**

In the late 1960s, Ivar Jacobson invented what later became known as use cases while working on telephony systems at Ericsson. In the late 1980s, he introduced them to the object-oriented programming community, where they were recognized as filling a significant gap in the requirements process. I took Jacobson’s course in the early 1990s. While neither he nor his team used my phrases *goal* and *goal failure*, it eventually became clear to me that they had been using these notions. In several comparisons, he

and I have found no significant contradictions between his and my models. I have slowly extended his model to accommodate recent insights.

I constructed the Actors and Goals conceptual model in 1994 while writing use case guides for the IBM Consulting Group. It explained away much of the mystery of use cases and provided guidance as to how to structure and write them. The Actors and Goals model has circulated informally since 1995 at <http://members.aol.com/acockburn> and later at [www.usecases.org](http://www.usecases.org), and finally appeared in the *Journal of Object-Oriented Programming* in 1997, in an article I authored entitled “Structuring Use Cases with Goals.”

From 1994 to 1999, the ideas stayed stable, even though there were a few loose ends in the theory. Finally, while teaching and coaching, I saw why people were having such a hard time with such a simple idea (never mind that I made many of the same mistakes in my first tries!). These insights, plus a few objections to the Actors and Goals model, led to the explanations in this book and to the Stakeholders and Interests model, which is a new idea presented here.

The Unified Modeling Language (UML) has had little impact on these ideas—and vice versa. Gunnar Overgaard, a former colleague of Jacobson’s, wrote most of the UML use case material and kept Jacobson’s heritage. However, the UML standards group has a strong drawing-tools influence, with the effect that the textual nature of use cases has been lost in the standard. Gunnar Overgaard and Ivar Jacobson discussed my ideas and assured me that most of what I have to say about a use case fits *within* one of the UML ellipses, and hence neither affects nor is affected by what the UML standard has to say. That means that you can use the ideas in this book quite compatibly with the UML 1.3 use case standard. On the other hand, if you only read the UML standard, which does not discuss the content or writing of a use case, you will not understand what a use case is or how to use it, and you will be led in the dangerous direction of thinking that use cases are a graphical, as opposed to a textual, construction. Since the goal of this book is to show you how to write effective use cases and the standard has little to say in that regard, I have isolated my remarks about UML to Appendix A.

## ***Samples Used***

The writing samples in this book were taken from live projects as much as possible, and they may seem slightly imperfect in some instances. I intend to show that they were sufficient to the needs of the project teams that wrote them, and that those imperfections are within the variations and economics permissible in use case writing.

The Addison-Wesley editing crew convinced me to tidy them up more than I originally intended, to emphasize correct appearance over the actual and adequate appearance. I hope you will find it useful to see these examples and recognize the

writing that happens on projects. You may apply some of my rules to these samples and find ways to improve them. That sort of thing happens all the time. Since improving one's writing is a never-ending task, I accept the challenge and any criticism.

## ***Use Cases in The Crystal Collection***

This is just one in a collection of books, The Crystal Collection for Software Professionals, that highlights lightweight, human-powered software development techniques. Some books discuss a single technique, some discuss a single role on a project, and some discuss team collaboration issues.

*Crystal* works from two basic principles:

- v Software development is a cooperative game of invention and communication. It improves as we develop people's personal skills and increase the team's collaboration effectiveness.
- v Different projects have different needs. Systems have different characteristics and are built by teams of differing sizes, with members having differing values and priorities. It is impossible to name one, best way of producing software.

The foundation book for the Crystal Collection, *Software Development as a Cooperative Game*, elaborates the ideas of software development as a cooperative game, of methodology as a coordination of culture, and of methodology families. That book separates the different aspects of methodologies, techniques and activities, work products and standards. The essence of the discussion, as needed for use cases, appears in this book in Section 1.2, Your Use Case Is Not My Use Case on page 7.

*Writing Effective Use Cases* is a technique guide, describing the nuts-and-bolts of use case writing. Although you can use the techniques on almost any project, the templates and writing standards must be selected according to each project's needs.

# Acknowledgments

---

Thanks to lots of people. Thanks to the people who reviewed this book in draft form and asked for clarification on topics that were causing their clients, colleagues, and students confusion. Special thanks to Russell Walters, a practiced person with a sharp eye for the direct and practical needs of the team, for his encouragement and very specific feedback. Thanks to FirePond and Fireman's Fund Insurance Company for the live use case samples. Pete McBreen, the first to try out the Stakeholders and Interests model, added his usual common sense, practiced eye, and suggestions for improvement. Thanks to the Silicon Valley Patterns Group for their careful reading of early drafts and their educated commentary on various papers and ideas. Mike Jones at the Fort Union Beans & Brew thought up the bolt icon for subsystem use cases.

Susan Lilly deserves special mention for the exact reading she did, correcting everything imaginable: sequencing, content, formatting, and even use case samples. The huge amount of work she contributed is reflected in the much improved final copy.

Other reviewers who contributed detailed comments and encouragement include Paul Ramney, Andy Pols, Martin Fowler, Karl Waclawek, Alan Williams, Brian Henderson-Sellers, Larry Constantine, and Russell Gold. The editors at Addison-Wesley did a good job of cleaning up my usual ungainly sentences and frequent typos.

Thanks to the people in my classes for helping me debug the ideas in the book.

Thanks again to my family, Deanna, Cameron, Sean, and Kieran, and to the people at the Fort Union Beans & Brew who once again provided lots of caffeine and a convivial atmosphere.

More on use cases is at the web sites I maintain: *members.aol.com/acockburn* and *www.usecases.org*. Just to save us some future embarrassment, my name is pronounced Cō-burn, with a long o.



*This page intentionally left blank*

*This page intentionally left blank*

## Chapter 3

---

# Scope

*Scope* is the word we use for the extent of what we design as opposed to someone else's design job or an already existing design.

Keeping track of the scope of a project, or even just the scope of a discussion, can be difficult. The consultant Rob Thomsett introduced me to a wonderful little tool for tracking and managing scope discussions—the *in/out list*. Absurdly simple and remarkably effective, it can be used to control scope discussions for ordinary meetings as well as project requirements.

Simply construct a table with three columns. The left column contains any topic; the next two columns are labeled “In” and “Out.” Whenever there might confusion as to whether a topic is within the scope of the discussion, add it to the table and ask people whether it is in or out. The amazing result, as Rob described and I have seen, is that while it is completely clear to each person in the room whether the topic is in or out, the views are often opposing. Rob relates that sometimes it requires an appeal to the project's steering committee to settle whether a particular topic really is within the scope of work or not. In or out can make a difference of many work-months. Try this technique on your next project or perhaps your next meeting.

Table 3.1 is a sample in/out list we produced for our purchase request tracking system.

Use the in/out list right at the beginning of the requirements or use case writing activity, to separate the things that are within the scope of work from those that are out of scope. Refer to it whenever the discussion seems to be going off track or some requirement is creeping into the discussion that might not belong. Update the chart as you go.

Use the in/out list for topics relating to both the functional scope and the design scope of the system under discussion.

**Table 3.1.** A Sample In/Out List

Topic	In	Out
Invoicing in any form		Out
Producing reports about requests (e.g., by vendor, by part, by person)	In	
Merging requests into one PO	In	
Partial deliveries, late deliveries, wrong deliveries	In	
All new system services, software	In	
Any nonsoftware parts of the system		Out
Identification of any preexisting software that can be used	In	
Requisitions	In	

### 3.1 FUNCTIONAL SCOPE

Functional scope refers to the services your system offers and that will eventually be captured by the use cases. As you start your project, however, it is quite likely that you won't know it precisely. You are deciding the functional scope at the same time you are identifying the use cases—the two tasks are intertwined. The in/out list helps with this, since it allows you to draw a boundary between what is in and what is out of scope. The other two tools are the *actor-goal list* and the *use case briefs*.

#### The Actor-Goal List

The actor-goal list names all the user goals that the system supports, showing the system's functional content. Unlike the in/out list, which shows items that are both in and out of scope, the actor-goal list includes only the services that will actually be supported by the system. Table 3.2 is one project's actor-goal list for the purchase request tracking system.

To make this list, construct a table of three columns. Put the names of the primary actors—the actors having the goals—in the left column; put each actor's goals with respect to the system in the middle column; and put the priority, or an initial guess as to the release in which the system will support that goal, in the third column. Update this list continually over the course of the project so that it always reflects the status of the system's functional boundary.

Some people add additional columns—*trigger*, to identify the use cases that will get triggered by time instead of by a person, and *business priority, development complexity*,

**Table 3.2.** *A Sample Actor-Goal List*

<b>Actor</b>	<b>Task-level Goal</b>	<b>Priority</b>
Any	Check on requests	1
Authorizer	Change authorizations	2
Buyer	Change vendor contacts	3
Requestor	Initiate a request	1
	Change a request	1
	Cancel a request	4
	Mark request delivered	4
	Refuse delivered goods	4
Approver	Complete request for submission	2
Buyer	Complete request for ordering	1
	Initiate PO with vendor	1
	Alert of nondelivery	4
Authorizer	Validate Approver's signature	3
Receiver	Register delivery	1

and *development priority*, so they can separate the business needs from the development costs to derive the development priority.

The actor-goal list is the initial negotiating point between the user representative, the financial sponsor, and the development group. It focuses the layout and content of the project.

### ***The Use Case Briefs***

I will keep repeating the importance of managing your energy and working at low levels of precision wherever possible. The actor-goal list is the lowest level of precision in describing system behavior, and it is very useful for working with the total picture of the system. The next level of precision will either be the main success scenario or a *use case brief*.

The use case brief is a two-to-six sentence description of use case behavior, mentioning only the most significant activity and failures. It reminds people of what is going on in the use case. It is useful for estimating work complexity. Teams constructing

**Table 3.3.** *Sample Use Case Briefs*

<b>Actor</b>	<b>Goal</b>	<b>Brief</b>
Production Staff	Modify the administrative area lattice	Production staff adds administrative area metadata (administrative hierarchy, currency, language code, street types, etc.) to the reference database. Contact information for source data is cataloged. This is a special case of updating reference data.
Production Staff	Prepare digital cartographic source data	Production staffs convert external digital data to a standard format and validate and correct it in preparation for merging with an operational database. The data is cataloged and stored in a digital source library.
Production and Field Staff	Commit update transactions of a shared check-out to an operational database	Staff applies accumulated update transactions to an operational database. Nonconflicting transactions are committed to the operational database. The application context is synchronized with the operational database. Committed transactions are cleared from the application context, leaving the operational database consistent, with conflicting transactions available for manual/interactive resolution.

from commercial, off-the-shelf components (COTS) use this description in selecting the components. Some project teams, such as those having extremely good internal communications and continual discussion with their users, never write more than these use case briefs for their requirements; they keep the rest of the requirements in the continual discussions, prototypes, and frequently delivered increments.

You can prepare the use case brief as a table, as an extension to the actor-goal list, or directly as part of the use case body in its first draft. Table 3.3 is a sample of briefs, thanks to Paul Ford, Steve Young, and Paul Bouzide of Navigation Technologies.

## **3.2 DESIGN SCOPE**

Design scope is the extent of the system—I would say “spatial extent” if software took up space. It is the set of systems, hardware and software, that we are charged with designing or discussing; it is that boundary. If we are to design an ATM, we are to produce hardware and software that sits in a box—the box and everything in it is ours to design. The computer network that the box will talk to is not ours to design—it is out of the design scope.

From now on, when I write *scope* alone, I mean *design scope*. This is because the functional scope is adequately defined by the actor-goal list and the use cases, while the design scope is a topic of concern in every use case.

As the following story illustrates, it is very important that the writer and reader are in agreement about the design scope for a use case—and correct. The price of being wrong can be a factor of two or more in cost, with disastrous results for the outcome of a contract. The readers of a use case must quickly see what you intend to be inside the system boundary. That will not be obvious just from the name of the use case or the primary actor. Systems of different sizes show up even within the same use case set.

Typically, writers consider the scope of the system to be so obvious that they don't mention it. However, once there are multiple writers and multiple readers, the design scope of a use case is not obvious at all. One writer is thinking of the entire corporation as the scope (see Figure 3.1), one is thinking of all of the company's software systems, one is thinking of the new, client–server system, and one is thinking of only the client or only the server. Readers, having no clue as to what is meant, get lost or misunderstand the document.

What can we do to clear up the misunderstanding?

The only answer I have found is to *label each and every use case with its design scope*, using specific names for the most significant scopes. To be concrete, let us suppose

### u A Short, True Story

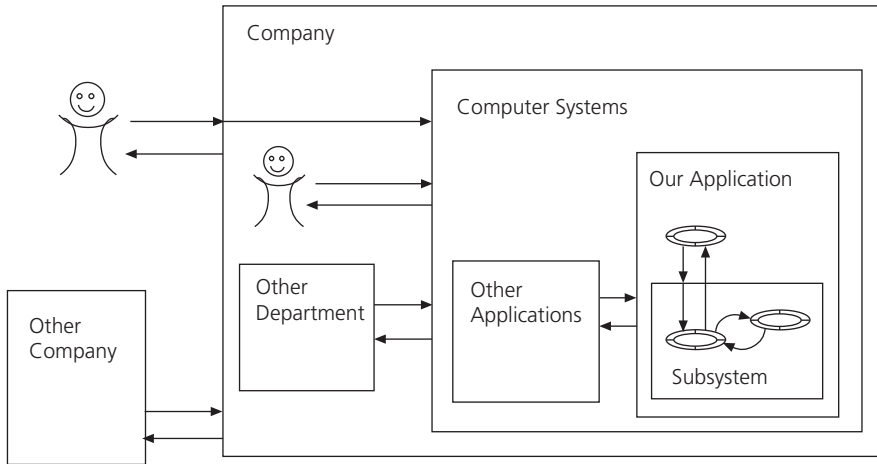
To help with constructing a fixed-time, fixed-cost bid of a large system, we were walking through some sample designs. I picked up the printer and spoke its function. The IS expert laughed. “You personal computer people crack me up,” he said, “You think we just use a little laser printer to print our invoices? We have a huge printing system, with a chain printer, batch I/O, and everything. We produce invoices by the boxful!”

I was shocked. “You mean the printer is not in the scope of the system?”

“Of course not! We'll use the printing system we already have.”

Indeed, we found that there was a complicated interface to the printing system. Our system was to prepare a magnetic tape with things to be printed. Overnight, the printing system would read the tape and print what it could. It would prepare a reply tape describing the results of the printing job, with error records for anything it couldn't print. The following day, our system would read back the results and note what had not been printed correctly. The design job for interfacing to that tape was significant, and completely different from what we had been expecting.

The printing system was not for us to design, but was for us to use. It was out of our design scope. (It was, as described in Section 3.3, a *supporting actor*.) Had we not detected this mistake, we would have written the use case to include it in our scope and turned in a bid to build more system than was needed.



**Figure 3.1** Design scope can be any size

that MyTelCo is designing a NewApp system, which includes a Searcher subsystem. The design scope names are these:

- v *Enterprise* (i.e., *MyTelCo*) 🏠. You are discussing the behavior of the entire organization or enterprise in delivering the goal of the primary actor. Label the *Scope* field of the use case with the name of the organization—*MyTelCo*—rather than just “the company.” If discussing a department, use the department name. Business use cases are written at the enterprise scope.
- v *System* (i.e., *NewApp*) 📦. This is the piece of hardware or software you are charged with building. Outside the system are all the pieces of hardware, software, and humanity that the system is to interface with.
- v *Subsystem* (i.e., *Searcher*) 🧩. You have opened up the main system and are about to talk about how a piece of it works.

### Using Graphical Icons to Highlight the Design Scope

Consider attaching a graphic to the left of the use case title to signal the design scope to readers before they start reading. There are no tools at this time to manage the icons, but I find that drawing them reduces confusion. In this book I label each use case with its appropriate icon to make it easier for you to note its scope.

As you read the following list, remember that a *black-box* use case does not discuss the internal structure of the system under discussion while a *white-box* use case does.



- A *business* use case has the enterprise as its scope. Its graphic is a building. Color it grey (🏠) if you treat the whole enterprise as a black box. Color it white (🏠) if you talk about the departments and staff within the organization.
- A *system* use case has a computer system as its scope. Its graphic is a box. Color it grey (📦) if you treat it as a black box, white (📦) if you reveal how its componentry works.
- A *component* use case is about a subsystem or component of the system under design. Its graphic is a bolt (🔩). See Use Cases 13 through 17 for an example.

## Design Scope Examples

I offer three examples to illustrate systems at different scopes.

### (1) Enterprise-to-System Scope

Suppose that we work for telephone company, *MyTelCo*, which is designing a new system, *Acura*, to take orders for services and upgrades. *Acura* consists of a workstation connected to a server. The server will be connected to a mainframe running the old system, *BSSO*. *BSSO* is just a terminal attached to the mainframe. We are not allowed to make any changes to it; we can only use its existing interfaces.

The primary actors for *Acura* include the customer, the clerk, various managers, and *BSSO* (we are clear that *BSSO* is not within our scope).

Let's find a few of the goals the system should support. The most obvious is "Add a new service." We decide that the primary actor for that is the company clerk, acting on behalf of the customer. We sit down to write a few use cases.

The immediate question is "What is the system under discussion?" It turns out that there are two that interest us:

- *MyTelCo*. We are interested in the question, "What does *MyTelCo*'s service look like to the customer, showing the new service implementation in its complete form, from initial request to implementation and delivery?" This question is of double interest. The company managers will want to see how the new system appears to the outside world, and the implementation team will want to see the context in which the new system will sit.

This use case will be written at the enterprise scope (🏠), with the Scope field labeled *MyTelCo* and the use case written without mention of company-internal players (no clerks, no departments, no computers). This sort of use case is often referred to as a *business use case*, since it is about the business.

- *Acura*. We are interested in the question, "How does *Acura*'s service appear, at its interface to the clerk or customer on one side and to the *BSSO* system on the

other side?” This is the use case the designers care most about, since it states exactly what they are to build. The use case will be written at the system scope (🏠), with the Scope field labeled “Acura.” It will freely mention clerks and departments and other computer systems, but not the workstation and the server subsystems.

We produce two use cases. To avoid having to repeat the same information twice, we write the enterprise use case at a higher level (the kite symbol), showing MyTelCo responding to the request, delivering it, and perhaps even charging for it and getting paid. The purpose of the enterprise use case is to show the context around the new system. Then we describe in detail the 5- to 20-minute handling of the request in the user-goal use case having Acura as its scope.

### Use Case 6 🏠 Add New Service (Enterprise) 🪁

---

**Primary Actor:** Customer

**Scope:** MyTelCo

**Level:** Summary

1. Customer calls MyTelCo, requests new service . . .
2. MyTelCo delivers . . . etc. . . .

### Use Case 7 🏠 Add New Service (Acura) 🪁

---

**Primary Actor:** Clerk for external customer

**Scope:** Acura

**Level:** User goal

1. Customer calls in, clerk discusses request with customer.
2. Clerk finds customer in Acura.
3. Acura presents customer’s current service package . . . etc. . . .

No use case will be written with a scope of Acura workstation or Acura server, as these are not of interest to us. Later, someone in the design team may choose to document Acura’s subsystem design using use cases. At that time, they will write two use cases, one with a scope of Acura workstation, the other with a scope of Acura server. My experience is that these use cases are never written, since there are other adequate techniques for documenting subsystem architecture.

## (2) Many Computers to One Application

The following is a less common situation, but one that is very difficult. Let us build onto the MyTelCo situation.

Acura will slowly replace BSSO. New service requests will be put into Acura and then modified using BSSO. Over time, Acura will take on more function. The two systems must co-exist and synchronize with each other. Thus, use cases have to be written for both systems: Acura being entirely new and BSSO being modified to synchronize with it.

The difficulty in this situation is that there are four use cases, two for Acura and two for BSSO. There is one use case for each system having the clerk as primary actor and one having the other computer system as the primary actor. There is no way to avoid these four use cases, but people looking at them get confused because they look redundant.

To document this situation, I first write a summary-level use case whose scope is both computer systems. This gives me a chance to document their interactions over time. In that use case, I reference the specific use cases that comprise each system's requirements. This first use case will be of the white-box type (note the white-box symbol).

The situation is complicated enough that I also include diagrams of each use case's scope.

### Use Case 8 Enter and Update Requests (Joint System)

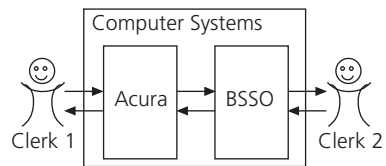
**Primary Actor:** Clerk for external customer

**Scope:** Computer systems, including Acura and BSSO (see diagram)

**Level:** Summary

**Main Success Scenario:**

1. Clerk adds new service into Acura.
2. Acura notes new service request in BSSO.
3. Some time later, Clerk updates service request in BSSO.
4. BSSO notes the updated request in Acura.



The four sub use cases are all user-goal use cases and get marked with the sea-level symbol. Although they are all system use cases, they are for different systems—hence the diagrams. In each diagram, I circle the primary actor and shade the SuD. The use cases are black-box this time, since they are requirements for new work. In

addition, I give them slightly different verb names, using the verb “note” to indicate one system synchronizing with the other.

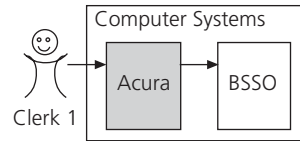
### Use Case 9 Add New Service (into Acura)

**Primary Actor:** Clerk for external customer

**Scope:** Acura

**Level:** User goal

... use case body follows. . .



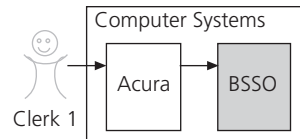
### Use Case 10 Note New Service Request (in BSSO)

**Primary Actor:** Acura

**Scope:** BSSO

**Level:** User goal

... use case body follows. . .



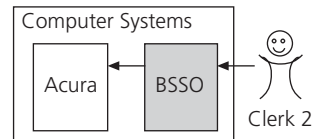
### Use Case 11 Update Service Request (in BSSO)

**Primary Actor:** Clerk for external customer

**Scope:** BSSO

**Level:** User goal

... use case body follows. . .



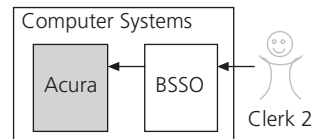
### Use Case 12 Note Updated Request (in Acura)

**Primary Actor:** BSSO

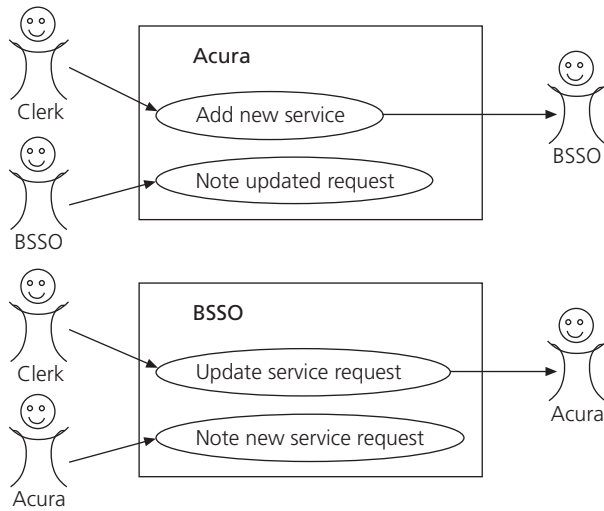
**Scope:** Acura

**Level:** User Goal

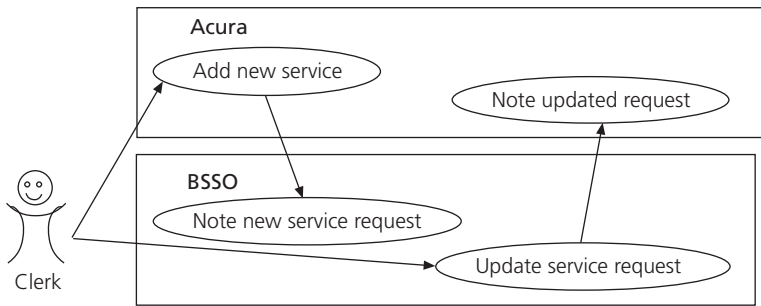
... use case body follows. . .



If you are using UML use case diagrams, you might draw the summary-level use case instead of writing it. That still does not reduce the confusion within the four user-goal use cases, so you should still carefully mark their primary actor, scope, and level, and possibly still draw the scope diagrams within the use cases.



**Figure 3.2 Use case diagrams for Acura–BSSO.** This is the UML style of denoting the interactions between the two systems. The upper section shows that BSSO is a supporting actor to one use case of Acura and a primary actor to another use case. In the lower diagram, the roles are reversed.



**Figure 3.3 A combined use case diagram for Acura–BSSO.** This drawing shows the relationships of the four use cases most clearly, but is nonstandard, since it shows one system’s use case triggering another system’s use case.

Personally, I do not find that this eliminates much confusion. I would consider drawing the nonstandard use case diagram in Figure 3.3 to show the connection between the two systems. This diagram is clearer but harder to maintain over time. Draw whichever you and your readers find communicates best for you.

### (3) Nuts and Bolts Use Cases

At the far end of the scale, let's look at the way one group documented their design framework with use cases. They started with an 18-page, diagram-loaded description of the rules for their framework. They decided it was too hard to read and experimented with use cases as the descriptive technique.

The group spent one week on the task. First they drafted 40 use cases to make sure they had captured all the requests their framework would handle. Using extensions and the data variations list, they revised those down to just six.

Most readers will find these use cases incomprehensible because they are not in that business. However, I expect some readers to be technical programmers looking for ways to document their designs, so I include these use cases to show how this group documented an internal architecture and how they made use of the variations list. I find them fairly easy to read, given the complexity of their problem. Notice that sub use cases are underlined. Thanks to Dale Margel in Calgary for the writing.

#### General Description:

The overall architecture must be able to handle concurrent tasks. To do this, it must support Process Threads and Resource Locking. These services are handled by the Concurrency Service Framework (CSF). CSF is used by client objects to protect critical sections of code from unsafe access by multiple processes.

### Use Case 13 Serialize Access to a Resource

**Primary Actor:** Service Client object

**Scope:** Concurrency Service Framework (CSF)

**Level:** User goal

**Main Success Scenario:**

1. Service Client asks a Resource Lock to give it specified access.
2. The Resource Lock returns control to the Service Client so that it may use the Resource.
3. Service Client uses the Resource.
4. Service Client informs the Resource Lock that it is finished with the Resource.
5. Resource Lock cleans up after the Service Client.

**Extensions:**

- 2a. Resource Lock finds that Service Client already has access to the resource:
  - 2a1. Resource Lock applies a lock conversion policy (Use Case 14) to the request.
- 2b. Resource Lock finds that the resource is already in use:
  - 2b1. The Resource Lock applies a compatibility policy (Use Case 15) to grant access to the Service Client.
- 2c. Resource Locking Holding time limit is nonzero:
  - 2c1. Resource Lock starts the holding timer.

- 3a. Holding Timer expires before the Client informs the Resource Lock that it is finished:
    - 3a1. Resource Lock sends an Exception to the Client's process.
    - 3a2. Fail!
  - 4a. Resource Lock finds nonzero lock count on Service Client:
    - 4a1. Resource Lock decrements the reference count of the request.
    - 4a2. Success!
  - 5a. Resource Lock finds that the resource is currently not in use:
    - 5a1. Resource Lock applies an access selection policy (Use Case 16) to grant access to any suspended service clients.
  - 5b. Holding Timer is still running:
    - 5b1. Resource Lock cancels Holding Timer.
- Technology and Data Variations List:**
- 1. The specified requested access can be:
    - ∪ For exclusive access
    - ∪ For shared access
  - 2c. The lock holding time-out can be specified by:
    - ∪ The Service Client
    - ∪ A Resource Locking policy
    - ∪ A global default value

## Use Case 14 Apply a Lock Conversion Policy

**Primary Actor:** Client object

**Scope:** Concurrency Service Framework (CSF)

**Level:** Subfunction

**Main Success Scenario:**

1. Resource Lock verifies that request is for exclusive access.
2. Resource Lock verifies that Service Client already has shared access.
3. Resource Lock verifies that there is no Service Client waiting to upgrade access.
4. Resource Lock verifies that there are no other Service Clients sharing the resource.
5. Resource Lock grants Service Client exclusive access to the resource.
6. Resource Lock increments Service Client lock count.

**Extensions:**

- 1a. Resource Lock finds that the request is for shared access:
  - 1a1. Resource Lock increments lock count on Service Client.
  - 1a2. Success!
- 2a. Resource Lock finds that the Service Client already has exclusive access:
  - 2a1. Resource Lock increments lock count on Service Client.
  - 2a2. Success!

- 3a. Resource Lock finds that there is another Service Client waiting to upgrade access:
  - 3a1. Signal Service Client that requested access could not be granted.
  - 3a2. Fail!
- 4a. Resource Lock finds that there are other Service Clients using the resource:
  - 4a1. Resource Lock makes Service Client wait for resource access (Use Case 17).

### Use Case 15 Apply an Access Compatibility Policy

**Primary Actor:** Service Client object

**Scope:** Concurrency Service Framework (CSF)

**Level:** Subfunction

**Main Success Scenario:**

1. Resource Lock verifies that request is for shared access.
2. Resource Lock verifies that all current usage of resource is for shared access.

**Extensions:**

- 2a. Resource Lock finds that the request is for exclusive access:
  - 2a1. Resource Lock makes Service Client wait for resource access (Use Case 17) (the process is resumed later by the Lock serving strategy).
- 2b. Resource Lock finds that the resource is being exclusively used:
  - 2b1. Resource Lock makes Service Client wait for resource access (Use Case 17)

**Variations:**

1. The compatibility criterion may be changed.

### Use Case 16 Apply an Access Selection Policy

**Primary Actor:** Client object

**Scope:** Concurrency Service Framework (CSF)

**Level:** Subfunction

**Main Success Scenario:**

**Goal in Context:** Resource Lock must determine which (if any) waiting requests should be served.

**Note:** This strategy is a point of variability.

1. Resource Lock selects oldest waiting request.
2. Resource Lock grants access to selected request(s) by making its process runnable.

**Extensions:**

- 1a. Resource Lock finds no waiting requests:
  - 1a1. Success!
- 1b. Resource Lock finds a request waiting to be upgraded from a shared to an exclusive access:
  - 1b1. Resource Lock selects the upgrading request.



1c. Resource Lock selects a request that is for shared access:

1c1. Resource repeats [Step 1] until the next one is for exclusive access.

**Variations:**

1. The selection ordering criterion may be changed.

## Use Case 17 Make Service Client Wait for Resource Access

**Primary Actor:** Client object

**Scope:** Concurrency Service Framework (CSF)

**Level:** Subfunction

**Main Success Scenario:**

**Used By:** CC 2,4 Resource Locking:

1. Resource Lock suspends Service Client process.

2. Service Client waits until resumed.

3. Service Client process is resumed.

**Extensions:**

1a. Resource Lock finds that a waiting time-out has been specified:

1a1. Resource Lock starts timer.

2a. Waiting Timer expires:

2a1. Signal Service Client that requested access could not be granted.

2a2. Fail!

**Technology and Data Variations List:**

1a1. The Lock waiting time-out can be specified by:

- ↳ The Service Client
- ↳ A Resource Locking policy
- ↳ A global default value

### 3.3 THE OUTERMOST USE CASES

In the Enterprise-to-System Scope subsection on page 41, I recommend writing two use cases, one for the system under design and one at an outer scope. Now we can get more specific about that: For each use case, find the outermost design scope at which it still applies and write a summary-level use case at that scope.

The use case is written to a design scope. Usually, you can find a wider design scope that still has the primary actor outside it. If you keep widening the scope, you reach the point at which widening it farther would bring the primary actor inside. That is the *outermost scope*. Sometimes the outermost scope is the enterprise, sometimes the department, and sometimes just the computer. Often, the computer department is the primary actor on computer security use cases, the marketing department

is the primary actor on advertising use cases, and the customer is the primary actor on the main system function use cases.

Typically, there are only two to five outermost use cases for the entire system, so not every use case gets written twice. There are so few of them because each one merges the primary actors having similar goals on the same design scope, and pulls together all the lower-level use cases for those actors.

I highly recommend writing the outermost use cases because it takes very little time and provides excellent context for the use case set. The outermost use cases show how the system ultimately benefits the most external users of the system; they also provide a table of contents for browsing through the system's behavior.

Let's visit the outermost use cases for MyTelCo and its Acura system.

MyTelCo decides to let web-based customers access Acura directly to reduce the load on the clerks. Acura will also report on the clerks' sales performance. Someone will have to set security access levels for customers and clerks. We have four use cases: *Add Service (by Customer)*, *Add Service (by Clerk)*, *Report Sales Performance*, and *Manage Security Access*.

We know we will have to write all four use cases with Acura as the scope of the SuD. We need to find the outermost scope for each of them.

The customer is clearly outside MyTelCo, so there is one outermost use case with the customer as primary actor and MyTelCo as scope. This use case will be at the summary level, showing MyTelCo as a black box, responding to the customer's request, delivering the service, and so on. In fact, the use case is outlined in Use Case 6, *Add New Service (Enterprise)*, on page 42.

The clerk is inside MyTelCo. The outermost scope for *Add Feature (by Staff)* is All Computer Systems. This use case will gather all the interactions the clerks have with the computer systems. I would expect all the clerks' user-goal use cases to be in this outermost use case, along with a few subfunction use cases, such as *Log In* and *Log Out*.

*Report Sales Performance* has the Marketing Department as the ultimate primary actor. The outermost use case is at scope Service Department and shows the Marketing Department interacting with All Computer Systems and the Service Department for setting up performance bonuses, reporting sales performance, and so on.

*Manage Security Access* has the Security or IT Department as its ultimate primary actor and either the IT Department or All Computer Systems as the outermost design scope. The use case references all the ways the Security Department uses All Computer Systems to set and track security issues.

Notice that these four outermost use cases cover security, marketing, service, and customers, using Acura in all the ways that it operates. It is unlikely that more

than these four need to be written for the Acura system, even if there are a hundred lower-level use cases to write.

### 3.4 USING THE SCOPE-DEFINING WORK PRODUCTS

You are defining the functional scope for your upcoming system, brainstorming, and moving between several work products on the whiteboard. On one part of the whiteboard, you have the in/out list to keep track of your scoping decisions (“No, Bob, we decided that a new printing system is out of scope—or do we need to revisit that entry in the in/out list?”). You have the actors and their goals in a list. You have a drawing of the design scope, showing the people, organizations, and systems that will interact with the system under discussion.

You find that you are evolving them all as you move between them, working out what you want your new system to do. You think you know what the design scope is, but a change in the in/out list moves the boundary. Now you have a new primary actor, and the goal list changes.

Sooner or later, you will probably find that you need a fourth item: a *vision statement* for the new system. The vision statement holds together the overall discussion. It helps you decide whether something should be in scope or out of scope in the first place.

When you are done, you have the four work products that bind the system’s scope:

- υ Vision statement
- υ Design scope drawing
- υ In/out list
- υ Actor-goal list

What I want you to take from this short discussion is that the four work products are intertwined and that you are likely to change them all while establishing the scope of the work to be done.

### 3.5 EXERCISES

#### ***Design Scope***

- 3.1. Name at least five system design scopes that the following user story fragment could be about: “. . . *Jenny is standing in front of her bank's ATM. It is dark. She has entered her PIN and is looking for the Enter button . . .*”
- 3.2. Draw a picture of the multiple scopes for an ATM, including hardware and software.

- 3.3. What system are you, personally, writing requirements for? What is its extent? What is inside it? What is outside it that it must communicate with? What is the system that encloses it, and what is outside that containing system that *it* must communicate with? Give the enclosing system a name.
- 3.4. Draw a picture of the multiple scopes for the Personal Advisors/Finance (PAF) system. (See Exercise 4.4.)
- 3.5. Draw a picture of the multiple scopes for a web application in which a user's workstation is connected through the web to your company's web server, which is attached to a legacy mainframe system.
- 3.6. Describe the difference between *enterprise-scope white-box business use cases* and *enterprise-scope black-box business use cases*.

# Index

---

- ! (user-goal use cases), 3–4, 6–7, 9–11, 62–64
- \* extensions, 103
- + (summary use cases), 3, 7, 62–67, 142, 144
- (subfunctions), 62–63, 66–67, 69, 142
- : extensions, 103
  
- Aas, Torfinn, 6
- Accuracy of use cases, 17
- Action steps, 90–98. *See also* Scenarios
  - bird's eye view for (Guideline 3), 91, 217
  - do until condition (Guideline 10), 96–97
  - exercises, 98, 249–250
  - extensions, 99–100
  - forward movement of (Guideline 4), 91–92
  - grammar (simple) for (Guideline 1), 90
  - intentions of actors (Guideline 5), 92–93
  - interface detail description, 92
  - numbering, 97, 218
  - reasonable set of (Guideline 6), 93–95
  - repeating steps, 96–97
  - scenarios, 88
  - sentence form for (Reminder 3), 206–207
  - systems interaction (Guideline 9), 96
  - timing (Guideline 8), 95–96
  - validation versus checking (Guideline 7), 95
  - "Who has the ball?" (Guideline 2, Reminder 5), 90–91, 207
- Actor-goal lists
  - case diagrams versus, 218
  - scope from, 36–37, 51
- Actor profile table, 58
  
- Actors, 54–60. *See also* Primary actors; Stakeholders; Supporting actors; System under discussion
  - aliases of, 58
  - design and primary actors, 56–57
  - exercises, 60, 246–247
  - goals conceptual model, xix, 23–29
  - internal actors and white-box cases, 59–60
  - offstage (tertiary, silent) actors, 30, 53–54
  - precision, 17
  - roles (Reminder 23), 57–58, 226
  - system delivery and primary actors, 57
  - system under discussion (SuD), 59
  - triggers, 54–55
  - ultimate primary actors, 54–55
  - Unified Modeling Language (UML), 58
  - use case production and primary actors, 55–56
  - use case writing and primary actors, 56–57
  - white-box cases and internal actors, 59–60
- Adding value with use cases, 15–16
- Adolph, Steve, 11–12, 133
- Agreement on use cases for completion, 142
- Aliases of actors, 58
- Alternate flows (extensions), 123
- Alternative paths, 217
- Anderson, Bruce, 243
- Arrow shapes in UML (Guideline 15), 236–237
- Asterisk (\*) for extensions, 103
- Atlantic Systems Guild, 13
  
- Bear, Kerry, 70
- Beck, Kent, 167, 187, 223–224

- Behavior. *See* Contract for behavior
- Bird's eye view (Guideline 3), 91, 217
- Black-box requirements, 217
- Black-box use cases (grey), 4–7, 9–11, 40–41
- Black/indigo, underwater fish/clam graphic, minus sign (subfunctions), 3, 7, 62–63, 66–67, 69, 142
- Blue, sea-level waves graphic, exclamation mark (user-goal use cases), 3–4, 6–7, 9–11, 62–64
- Body of scenarios, 89
- Bolt graphic (component use cases), 41, 46–49
- Bouzide, Paul, 38
- Box graphic, grey/white (system use cases), 3–7, 9–11, 41, 157–159, 216
- Brainstorming
  - extensions, 101–104, 110
  - use cases for, 12, 16
- Bramble, Paul, 128
- Branch-and-join process for project planning, 180–183
- Building graphic, grey/white (business use cases), 3, 7, 41
- Business
  - priority, scope, 36
  - process to technology, 155–157
  - rules discovered by extensions, 100
  - setting and formats, 129
  - system use cases versus (Reminder 13), 216
  - use cases (building graphic, grey/white), 3, 7, 41
- Business process modeling, 153–160
  - business process to technology, 155–157
  - core business, working from, 154–155
  - designing versus modeling, 153–157
  - external primary actors, 154
  - linking business and system use cases, 157–159
  - modeling versus designing, 153–157
  - prior to use cases, 218
  - services, 154
  - stakeholders, 154
  - standard, 132, 134
  - system use cases, linking to business, 157–159
  - technology to business process, 157
  - triggers, 154
  - usage experts, 156–157
  - use case examples, 153
- Case diagrams versus actor-goal lists, 218
- CASE tools, 127, 227, 230
- Casual use cases, 7–9, 97, 120, 218
- Central Bank of Norway, 6
- Chrysler Comprehensive Compensation, 223
- Clam graphic, indigo/black, minus sign (subfunctions), 3, 7, 62–63, 66–67, 69, 142
- Cloud/kite graphic, white, plus sign (summary use cases), 3, 7, 62–67, 142, 144
- Clusters of use cases, 143–144
- Collaboration diagrams (UML) versus white-box cases, 218
- Colaizzi, John, 102, 145
- Collecting
  - scenarios, 27–29
  - use cases from large groups, 184–186
- Colon (:) for extensions, 103
- Completeness and formats, 131
- Completion of use cases, 141–142. *See also* Project planning
- Complexity and formats, 130–131
- Component use cases (bolt graphic), 41, 46–49
- Compound interactions, 25–27
- Conditions
  - extensions, 99–106
  - failure conditions (fourth work step), 16–17, 222
  - preconditions (Reminder 10), 2, 81–83, 211
  - scenarios, 88
- Conflict and formats, 131
- Consistency and formats, 130
- Constantine, Larry, 58, 92, 122, 163, 177
- Content and purpose misalignment, 193
- Context diagrams, 128, 227–228
- Contract for behavior, 23–33. *See also* Action steps; Goal levels; Scenarios; Reminders actors and goals conceptual model, xix, 23–29 compound interactions, 25–27 ever-unfolding story (Reminder 12), 26, 62, 215

- failure scenario, 31
- goal failures and responses, 25
- graphical model, 31–33, 229
- interaction between two actors, 31
- interactions, compound, 25–27
- internal state change, 31
- main success scenarios (third work step), 3, 17, 28, 87–89, 222
- offstage actors, 30, 53–54
- partial ordering, 26
- primary actors, 23, 27, 30–31
- scenario collection, 27–29
- scenarios, 25
- sequences of interactions, 25–27
- sets of possible sequences, 26–27
- stakeholders and interests conceptual model, 29–31
- striped trousers image, 27–29
- subgoals, 23–24
- supporting actors, 23–24
- system under discussion (SuD), 24–25, 29
- Unified Modeling Language (UML), 31
- validation to protect stakeholders, 31
- Conversations, formats, 122
- Coppolo, Dawn, 70
- Core business, working from, 154–155
- Core values and variations (Reminder 14), 216–219
- Coverage and formats, 130
- Create, Retrieve, Update, Delete (CRUD) use cases, 145–150
- Cross-linking from use cases to missing requirements, 164–165
- Cultures and formats, 129
- Curran, Eileen, 70
- Data
  - field details and checks (sixth and seventh work steps), 223
  - requirement precision, 162–164
  - and technology variations, 111–112
- Delivery and scenarios, 170
- Design
  - modeling versus, 153–157
  - primary actors and, 56–57
  - project planning, 171–174
  - scenarios and, 177
  - use cases for, 174–177
- Design scope. *See* Scope
- Detailed functional requirements standard, 132, 137
- Development
  - complexity, scope, 36
  - priority, scope, 37
  - team for scaling up, 144
- Diagram style, 127
- Dive-and-surface approach, 12
- Do until condition (Guideline 10), 96–97
- Documenting requirements from use cases, 12. *See also* Requirements
- Domain concepts from use cases, 177
- DOORS, 230
- Drawing use case diagrams, UML, 242–243
- Elementary business process, 68
- Ellipses and stick figures, UML, 233–234
- Emphasizing extensions, 103
- Empower IT, 145
- End condition, scenarios, 88
- Endings (two) of use cases (Reminder 8), 209–210
- Energy management, 16–17, 217, 221–223
- Enterprise-to-system scope, 41–42
- Essential use cases, 122
- Evans, Eric, 70
- Ever-unfolding story (Reminder 12), 26, 62, 215
- Exclamation mark (!) user-goal use cases, 3–4, 6–7, 9–11, 62–64
- Experience and formats, 130
- Extend relations, UML, 235–238
- Extension points, UML, 237–238
- Extensions, 99–110. *See also* Scenarios
  - action steps, 99–100
  - asterisk (\*) for, 103
  - brainstorming, 101–104, 110
  - business rules discovered, 100
  - colon (:) for, 103
  - conditions, 99–106
  - defined, 3
  - drawing lower in UML (Guideline 14), 236

- Extensions (*cont.*)
  - emphasizing, 103
  - exercises, 110, 251–252
  - failures within failures, 109
  - goal delivery, 100
  - handling, 106–110
  - indenting condition handling (Guideline 12), 108
  - linking, 114–117
  - merging conditions, 105–106
  - Rational Unified Process (RUP), 108
  - rationalizing, 104–105
  - rollup failures, 105–106
  - system detection of condition (Guideline 11), 102–103
  - use case creation from, 109–110
- External primary actors, 154
- eXtreme Programming (XP), 187, 223–224
- Failure. *See also* Extensions
  - conditions (fourth work step), 16–17, 222
  - handling (Reminder 21), 17, 225
  - scenario, 31
- Feature lists from use cases, 171–174
- Field details and field checks, 163–164
- Field lists, 163
- Finding right goal levels (Reminder 6), 68–69, 208
- Fireman’s Fund Insurance, 70–79
- FirePond Corporation, 158–159, 194, 205
- Fish/clam graphic, indigo/black, minus sign (subfunctions), 3, 7, 62–63, 66–67, 69, 142
- Ford, Paul, 38
- Form of use cases, 1
- Formality and formats, 130
- Formats, 119–138
  - alternate flows (extensions), 123
  - business process modeling standard, 132, 134
  - business setting and, 129
  - CASE tools, 127, 227, 230
  - casual, 7–9, 97, 120, 218
  - completeness and, 131
  - complexity and, 130–131
  - conflict and, 131
  - consistency and, 130
  - context diagrams, 128, 227–228
  - conversations, 122
  - coverage and, 130
  - cultures and, 129
  - detailed functional requirements standard, 132, 137
  - essential use cases, 122
  - exercises, 138, 252
  - experience and, 130
  - forces affecting writing styles, 128–132
  - formality and, 130
  - fully dressed, 4–11, 97, 119–120, 218
  - goals versus tasks, 131
  - graphical notations (Reminder 24), 127–128, 227–228
  - if-statement style, 126, 138, 218
  - Occam style, 126–127
  - one-column tables, 121
  - Rational Unified Process (RUP), 123–126
  - requirements elicitation standard, 132–133
  - resources and, 131
  - short, high-pressure project standard, 132, 136
  - sizing requirements standard, 132, 135
  - social interaction and, 129
  - stakeholder needs and, 129
  - standards for, 132–137
  - tables, 121–122
  - tasks versus goals, 131
  - two-column tables, 122
  - understanding level and, 129
  - Unified Modeling Language (UML), 128
  - Use Case 24 (Fully Dressed Use Case Template), 119–120
  - Use Case 25 (Actually Login, Casual Version), 120, 135, 218
  - Use Case 26 (Register for Courses), 124–126
  - Use Case 27 (Elicitation Template—Oble a New Biscum), 133, 250
  - Use Case 28 (Business Process Template—Symp a Carstromming), 134, 251–252
  - Use Case 29 (Sizing Template—Burble the Tramlng), 135, 252
  - Use Case 30 (High-Pressure Template: Kree a Ranfath), 136



- Use Case 31 (Use Case Name—Nathorize a Permion), 137
- Forward movement of action steps (Guideline 4), 91–92
- Fowler, Martin, 236
- Fully dressed use cases, 4–11, 97, 119–120, 218
- Functional decomposition of use cases, 176
- Functional scope, 36–38
  
- Generalizes relations, UML, 236, 239–241
- Goal-based core value, 216
- Goal levels, 61–79
  - actors conceptual model, xix, 23–29
  - delivery, extensions, 100
  - drawing higher in UML (Guideline 16), 240
  - elementary business process, 68
  - ever-unfolding story (Reminder 12), 26, 62, 215
  - exercises, 79, 247–248
  - failures and responses, 25
  - finding right (Reminder 6), 68–69, 208
  - graphical icons for, 67–68
  - length of use cases (Reminder 20), 69, 224
  - low, mistake, 192–193
  - outermost scope use cases, 49–51, 65–66, 215
  - precision, 17
  - raising and lowering, 69, 91–92, 192–193
  - scenarios, 88
  - second work step, 221–222
  - subfunctions (underwater fish/clam graphic, indigo/black, minus sign), 62–63, 66–67, 69, 142
  - summary (strategic) level (cloud/kite graphic, white, plus sign), 3, 7, 62–67, 142, 144
  - tasks format versus, 131
  - Use Case 18 (Operate an Insurance Policy), 65, 67, 153
  - Use Case 19 (Handle a Claim, Business), 59, 70–71, 153, 159
  - Use Case 20 (Evaluate Work Comp Claim), 65, 71–72, 153, 209
  - Use Case 21 (Handle a Claim, Systems), 65, 73–74, 156–157
  - Use Case 22 (Register a Loss), 74–78, 89, 109, 127, 209
  - Use Case 23 (Find a Whatever, Problem Statement), 66, 78–79, 150
  - user-goal level (sea-level waves graphic, blue, exclamation mark), 62–64
- Grammar (simple) for action steps (Guideline 1), 90
- Graphical icons/model. *See also* Unified Modeling Language
  - contract for behavior, 31–33, 229
  - for goal levels, 67–68
  - notations (Reminder 24), 127–128, 227–228
  - scope of, 45, 51
  - for scope, 40–41
- Graphical user interfaces (GUIs), keeping out (Reminder 7), 209, 219
- Greenberg, Marc, 70
- Grey (black-box use cases), 4–7, 9–11, 40–41
- Grey/white, box graphic (system use cases), 3–7, 9–11, 41, 157–159, 216
- Grey/white, building graphic (business use cases), 3, 7, 41
- Guarantees for stakeholders (Reminder 9), 2, 83–85, 210–211, 248
- Guidelines
  - arrow shapes in UML (Guideline 15), 236–237
  - bird's eye view (Guideline 3), 91, 217
  - detection of condition (Guideline 11), 102–103
  - do until condition (Guideline 10), 96–97
  - extension drawing lower in UML (Guideline 14), 236
  - forward movement of action steps (Guideline 4), 91–92
  - goals drawing higher in UML (Guideline 16), 240
  - grammar (simple) for action steps (Guideline 1), 90
  - higher-level goals in UML (Guideline 13), 235
  - indenting condition handling (Guideline 12), 108
  - intensions of actors (Guideline 5), 92–93
  - reasonable set of actions steps (Guideline 6), 93–95

- Guidelines (*cont.*)
- supporting actors on right in UML (Guideline 18), 243
  - systems interaction (Guideline 9), 96
  - timing (Guideline 8), 95–96
  - user goals in context diagram (Guideline 17), 243
  - validation versus checking (Guideline 7), 95
  - “Who has the ball?” (Guideline 2, Reminder 5), 90–91, 207
- GUIs, keeping out (Reminder 7), 209, 219
- Hammer, Michael, 154
- Handling extensions, 106–110
- Heymer, Volker, 108
- High-precision view of system’s functions, 180, 182–183
- High-pressure project standard, 132, 136
- Higher-level goals in UML (Guideline 13), 235
- Hoare, Tony, 126–127
- Hohmann, Luke, 163, 177
- Holistic Diversity pattern, 224
- “Hub-and-Spoke” requirements model, 15, 164
- Hunt, Andy, 227
- Hupp, Brent, 70
- IBM, xix, 180, 243
- If-statement style format, 126, 138, 218
- In/out list for scope, 35–36, 51
- Includes relations
- sub use cases (Reminder 4), 207
  - Unified Modeling Language (UML), 234–236
- Indenting condition handling (Guideline 12), 108
- Indigo/black, underwater fish/clam graphic, minus sign (subfunctions), 3, 7, 62–63, 66–67, 69, 142
- Information nicknames, 162
- Intensions of actors (Guideline 5), 92–93
- Interaction between two actors, 31
- Interactions, compound, 25–27
- Interface detail description, 92
- Internal actors and white-box cases, 59–60
- Internal state change, 31
- Italics for linking use cases, 113
- Ivey, Paula, 70
- Jacobson, Ivar, 93, 227
- Jewell, Nancy, 70
- Job titles (Reminder 22), 225–226
- Kite/cloud graphic, white, plus sign (summary use cases), 3, 7, 62–67, 142, 144
- Kraus, Andy, 180, 184–186
- Lazar, Nicole, 70
- Length of use cases (Reminder 20), 69, 224
- Level of view, 2, 7
- Lilly, Susan, 116, 145, 216
- Linking use cases, 113–117
- business and system use cases, 157–159
  - exercises, 117
  - extension use cases, 114–117
  - italics for, 113
  - sub use cases, 113
  - underlining for, 3, 113
  - Unified Modeling Language (UML) for, 114–115
- Lockwood, Lucy, 58, 92, 122, 163, 177
- Lotus Notes for tool, 229
- Low-precision
- representation of use cases, 143
  - view of system’s functions, 180–182
- Magdaleno, Trisha, 70
- Main success scenarios (third work step), 3, 17, 28, 87–89, 222
- Many computers to one application, 43–45
- Many use cases, 143–144
- Margel, Dale, 46
- Maxwell, Allen, 145
- McBreen, Pete, 178–180, 211
- Merging conditions, extensions, 105–106
- Minimal guarantees, 83–85, 248
- Minus sign (–) subfunctions, 62–63, 66–67, 69, 142
- Missing requirements, 161–165
- cross-linking from use cases to, 164–165
  - data requirement precision, 162–164
  - field details and field checks, 163–164
  - field lists, 163
  - “Hub-and-Spoke” requirements model, 15, 164

- information nicknames, 162
- precision in data requirements, 162–164
- Mistakes, costs of (Reminder 19), 223–224
- Mistakes fixed, 189–202
  - content and purpose misalignment, 193
  - goal levels, very low, 192–193
  - primary actors (none), 190–191
  - purpose and content misalignment, 193
  - system (none), 189–190
  - Use Case 36 (Research a Solution—Before), 122, 194–199, 205
  - Use Case 37 (Research Possible Solutions—After), 199–202
  - user interface details, too many, 191–192, 194–202
- Modeling versus designing, 153–157
- MSS. *See* Main success scenarios
- Navigation Technologies, 38
- Numbering actions steps, 97, 218
- Nuts and bolts use cases, 41, 46–49
- Object-oriented design from use cases, 176–177
- Occam language, 126–127
- Offstage actors, 30, 53–54
- One-column table format, 121
- Outermost scope use cases, 49–51, 65–66, 215
- Packages, UML, 143
- Paragraphs versus numbered steps, 97, 218
- Parameterized use cases, 150–151
- Partial ordering, 26
- Pass/fail tests (Reminder 10), 211–213
- Passini, Susan, 70
- Planning from use cases (Reminder 26), 167–170, 230
- Plus sign (+) summary use cases, 3, 7, 62–67, 142, 144
- Pratt, Pamela, 70
- Precision
  - data requirements, 162–164
  - system's functions, 180–183
  - use cases, 16–17
  - user interface (UI) design, 178
- Preconditions (Reminder 10), 2, 81–83, 211
- Primary actors, 54–58. *See also* Stakeholders; System under discussion
  - actor profile table, 58
  - aliases of, 58
  - contract for behavior, 23, 27, 30–31
  - defined, 1–2, 54
  - design and, 56–57
  - first work step, 221–222
  - none mistake, 190–191
  - roles versus (Reminder 23), 57–58, 226
  - scaling up using, 144
  - system delivery and, 57
  - triggers and, 54–55
  - ultimate primary actors, 54–55
  - Unified Modeling Language (UML), 58
  - use case production and, 55–56
  - use case writing and, 56–57
  - user goals for completion, 141
- Profile table, actor, 58
- Project-linking software, use cases as, 14–15
- Project planning, 167–186
  - branch-and-join process for, 180–183
  - collecting use cases from large groups, 184–186
  - “completeness,” 175
  - design documents and, 171–174
  - design from use cases, 174–177
  - domain concepts from use cases, 177
  - feature lists from use cases, 171–174
  - functional decomposition of use cases, 176
  - high-precision view of system's functions, 180, 182–183
  - low-precision view of system's functions, 180–182
  - object-oriented design from use cases, 176–177
  - precision of user interface (UI) design, 178
  - releases and use cases, 169–170, 230
  - Responsibility-Driven Design, 177
  - scenarios and design, 177
  - scenarios (complete) delivery, 170
  - task lists from use cases, 171–174
  - test cases from, 178–180
  - time required per use case, 184
  - Use Case 34 (Capture Trade-In), 172–173

Project planning (*cont.*)

- Use Case 35 (Order Goods, Generate Invoice, Testing Example), 178–179
  - use cases for (Reminder 26), 167–170, 230
  - user interface design from use cases, 177–178
  - writing, 180–186
- Prose essay, use cases as (Reminder 1), 205
- Purpose(s)
- and content misalignment, 193
  - of uses cases, 217
  - and writing style, 7–11
- Quality questions (Reminder 15), 11, 219
- Raising and lowering goal levels, 69, 91–92, 192–193
- Raison d'être (elementary business process), 68
- Rational Software Corporation, 123
- Rational Unified Process (RUP)
- extensions, 108
  - formats, 123–126
- Rationalizing extensions, 104–105
- Readability of use cases (Reminder 2), 205–206, 217
- Reasonable set of actions steps (Guideline 6), 93–95
- Recovery (fifth work step), 222–223
- Reference to another use case (underlined), 3
- Relational databases, 229
- Releases and use cases, 169–170, 230
- Releases for scaling up, 144
- Reminders, 205–230
- actor-goal lists versus case diagrams, 218
  - actors play roles (Reminder 23), 57–58, 226
  - alternative paths, 217
  - bird's eye view (Guideline 3), 91, 217
  - black-box requirements, 217
  - business process modeling, prior to use cases, 218
  - business versus system use cases (Reminder 13), 216
  - case diagrams versus actor-goal lists, 218
  - CASE tools, 127, 227, 230
  - casual versus fully dressed, 218
  - collaboration diagrams (UML) versus white-box cases, 218
  - core values and variations (Reminder 14), 216–219
  - data field details and checks (seventh work step), 223
  - data fields (sixth work step), 223
  - endings (two) of use cases (Reminder 8), 209–210
  - energy management, 16–17, 217, 221–223
  - ever-unfolding story (Reminder 12), 26, 62, 215
  - failure conditions (fourth work step), 16–17, 222
  - failure handling (Reminder 21), 17, 225
  - fully dressed versus casual, 218
  - goal-based core value, 216
  - goal level, getting right (Reminder 6), 68–69, 208
  - goals (second work step), 221–222
  - graphical notations (Reminder 24), 127–128, 227–228
  - guarantees for stakeholders (Reminder 9), 2, 83–85, 210–211, 248
  - GUIs, keeping out (Reminder 7), 209, 219
  - Holistic Diversity pattern, 224
  - if-statement style, 126, 138, 218
  - includes relation, sub use cases (Reminder 4), 207
  - job titles (Reminder 22), 225–226
  - length of use cases (Reminder 20), 69, 224
  - Lotus Notes for tool, 229
  - main success scenarios (third work step), 3, 17, 28, 87–89, 222
  - mistakes, costs of (Reminder 19), 223–224
  - numbered steps versus paragraphs, 97, 218
  - paragraphs versus numbered steps, 97, 218
  - pass/fail tests (Reminder 10), 211–213
  - preconditions (Reminder 10), 2, 81–83, 211
  - primary actors (first work step), 221–222
  - project planning, 167–170, 230 (Reminder 26)
  - prose essay, use cases as (Reminder 1), 205
  - purposes (several) of uses cases, 217
  - quality questions (Reminder 15), 11, 219

- readability of use cases (Reminder 2), 205–206, 217
- recovery (fifth work step), 222–223
- relational databases for, 229
- releases and use cases, 169–170, 230
- requirements and use cases (Reminder 16), 13–15, 19, 221
- requirements management tools, 230
- roles and actors (Reminder 23), 57–58, 226
- sentence form (one) (Reminder 3), 206–207
- sequence diagrams as use case text, 219
- stakeholders need guarantees (Reminder 9), 2, 83–85, 210–211, 248
- sub use cases, includes relation (Reminder 4), 207
- system versus business use cases (Reminder 13), 216
- tool debate (Reminder 25), 229–230
- 12-step recipe (Reminder 18), 223
- white-box cases versus collaboration diagrams (UML), 218
- “Who has the ball?” (Guideline 2, Reminder 5), 90–91, 207
- word processors with hyperlinks for tool, 229
- work breadth first (Reminder 17), 221–223
- Repeating actions steps, 96–97
- Requirements. *See also* Missing requirements; Use cases
  - discovery, 11–12, 133
  - elicitation standard, 132–133
  - management tools, 230
  - sizing requirements standard, 132, 135
  - use cases and (Reminder 16), 13–15, 19, 221
- RequisitePro, 230
- Resources and formats, 131
- Responsibility-Driven Design, 177
- Robertson, James, 13
- Robertson, Suzanne, 13
- Roles and actors (Reminder 23), 57–58, 226
- Rollup failures, extensions, 105–106
- Roshi, 211
- RUP. *See* Rational Unified Process
- Sampson, Steve, 70
- Sawyer, Jim, 8
- Scaling up, 143–144
- Scenarios. *See also* Action steps; Extensions
  - body, 89
  - collection, 27–29
  - condition for, 88
  - contract for behavior, 25
  - defined, 1, 3
  - delivery and, 170
  - design and, 177
  - end condition, 88
  - extensions, 88
  - goal to achieve, 88
  - main success scenarios, 3, 17, 28, 87–89, 222
- Schick Tanz, Jill, 70
- Scope (design scope), 35–52
  - actor-goal list for, 36–37, 51
  - business priority, 36
  - defined, 2
  - development complexity, 36
  - development priority, 37
  - enterprise-to-system scope, 41–42
  - exercises, 51–52, 245
  - functions scope, 36–38
  - graphical icons for, 40–41
  - graphical model, 45, 51
  - in/out list for, 35–36, 51
  - many computers to one application, 43–45
  - nuts and bolts use cases, 41, 46–49
  - outermost scope use cases, 49–51, 65–66, 215
  - triggers, 36
- Unified Modeling Language (UML), 44–45
  - use case briefs for, 37–38, 187
- Use Case 6 (Add New Service, Enterprise), 42, 50
- Use Case 7 (Add New Service, Acura), 42
- Use Case 8 (Enter and Update Requests, Joint System), 43, 59
- Use Case 9 (Add New Service into Acura), 44
- Use Case 10 (Note New Service Requests in BSSO), 44
- Use Case 11 (Update Service Request in BSSO), 44
- Use Case 12 (Note Updated Request in Acura), 44
- Use Case 13 (Serialize Access to a Resource), 46–47, 112

- Scope (*cont.*)
- Use Case 14 (Apply a Lock Conversion Policy), 47–48
  - Use Case 15 (Apply an Access Compatibility Policy), 48
  - Use Case 16 (Apply an Access Selection Policy), 48–49
  - Use Case 17 (Make Service Client Wait for Resource Access), 49
  - vision statements, 51
- Scott, Dave, 194–202
- Sea-level waves graphic, blue, exclamation mark (user-goal use cases), 3–4, 6–7, 9–11, 62–64
- Secondary actors. *See* Supporting actors
- Sentence form for action steps (Reminder 3), 206–207
- Sequence diagrams as use case text, 219
- Sequences of interactions, 25–27
- Services, business process modeling, 154
- Sets of possible sequences, 26–27
- Short, high-pressure project standard, 132, 136
- Silent (offstage) actors, 30, 53–54
- Situations for use cases, 7–11
- Sizing requirements standard, 132, 135
- Social interaction and formats, 129
- Software Futures CCH, 108
- S.R.A., 116, 145
- Stakeholders. *See also* Actors
  - business process modeling, 154
  - defined, 1–2, 53–54
  - interests conceptual model, 29–31
  - need guarantees (Reminder 9), 2, 83–85, 210–211, 248
  - needs and formats, 129
- Standards, 11, 132–137
- Statements, vision, 51
- Steps. *See* Action steps
- Strategic use cases (cloud/kite graphic, white, plus sign), 3, 7, 62–67, 142, 144
- Striped trousers image, contract for behavior, 27–29
- Subforms of use cases, 7–11
- Subfunctions (underwater fish/clam graphic, indigo/black, minus sign), 3, 7, 62–63, 66–67, 69, 142
- Subgoals, contract for behavior, 23–24
- Subject area for scaling up, 144
- Subordinate versus sub use cases, UML, 242
- Sub use cases
  - includes relation (Reminder 4), 207
  - linking, 113
- Success guarantees, 84–85, 248
- SuD. *See* System under discussion
- “Sufficient” use cases, 5
- Summary use cases (cloud/kite graphic, white, plus sign), 3, 7, 62–67, 142, 144
- Supporting (secondary) actors
  - contract for behavior, 23–24
  - defined, 59
  - on right in UML (Guideline 18), 243
- Swift, Jonathan, 24
- System under discussion (SuD)
  - actors, 59
  - contract for behavior, 24–25, 29
  - defined, 2
  - delivery and primary actors, 57
  - detection of condition (Guideline 11), 102–103
  - none mistake, 189–190
- System use cases (box graphic, grey/white), 3–7, 9–11, 41, 157–159, 216
- System versus business use cases (Reminder 13), 216
- Systems interaction (Guideline 9), 96
- Table format, 121–122
- Task lists from use cases, 171–174
- Tasks versus goals format, 131
- Technology and data variations, 111–112
- Technology to business process, 157
- Template for use cases, 7
- Tertiary (offstage) actors, 30, 53–54
- Test cases from project planning, 178–180
- Text-based use cases versus UML, 243
- Thomas, Dave, 227
- Thomsett, Rob, 35
- Time required per use case, 184
- Timing (Guideline 8), 95–96
- Tool debate (Reminder 25), 229–230
- Tools, CASE, 127, 227, 230
- Triggers
  - actors and, 54–55

- business process modeling, 154
- completion and, 141–142
- defined, 84–85
- scope and, 36
- 12-step recipe (Reminder 18), 223
- Two-column table format, 122
- UI. *See* User interface
- Ultimate primary actors, 54–55
- UML. *See* Unified Modeling Language
- Underlining for linking use cases, 3, 113
- Understanding level and formats, 129
- Underwater fish/clam graphic, indigo/black, minus sign (subfunctions), 3, 7, 62–63, 66–67, 69, 142
- Unified Modeling Language (UML), 233–243
  - actors, 58
  - arrow shapes, using different (Guideline 15), 236–237
  - collaboration diagrams versus white-box cases, 218
  - contract for behavior, 31
  - drawing use case diagrams, 242–243
  - ellipses and stick figures, 233–234
  - extend relations, 235–238
  - extension points, 237–238
  - extension use cases, 114–115
  - extension use cases, drawing lower (Guideline 14), 236
  - formats, 128
  - general goals, drawing higher (Guideline 16), 240
  - generalizes relations, 236, 239–241
  - higher-level goals (Guideline 13), 235
  - includes relations, 234–236
  - linking use cases, 114–115
  - packages, 143
  - scope, 44–45
  - subordinate versus sub use cases, 242
  - supporting actors on right (Guideline 18), 243
  - text-based use cases versus, 243
  - user goals in context diagram (Guideline 17), 243
- Usage experts, 156–157
- Usage narratives, 17–19
- Use case briefs for scope, 37–38, 187
- Use cases, 1–19. *See also* Action steps; Actors; Formats; Linking use cases; Project planning; Requirements; Scope (design scope)
  - accuracy, 17
  - adding value with, 15–16
  - black-box use cases (grey), 4–7, 9–11, 40–41
  - brainstorming using, 12
  - brainstorming with, 16
  - business use cases (building graphic, grey/white), 3, 7, 41
  - casual use cases, 7–9, 97, 120, 218
  - completion, 141–142
  - component use cases (bolt graphic), 41, 46–49
  - Create, Retrieve, Update, Delete (CRUD) use cases, 145–150
  - defined, 1–3
  - dive-and-surface approach using, 12
  - documenting requirements using, 12
  - energy management, 16–17, 217, 221–223
  - eXtreme Programming (XP), 187, 223–224
  - failure conditions, 16–17
  - failure handling (Reminder 21), 17, 225
  - form of, 1
  - fully dressed use cases, 4–11, 97, 119–120, 218
  - guarantees (Reminder 9), 2, 83–85, 210–211, 248
  - “Hub-and-Spoke” requirements model, 15, 164
  - length of (Reminder 20), 69, 224
  - level of view, 2, 7
  - main success scenarios (MSS), 3, 17, 28, 87–89, 222
  - parameterized use cases, 150–151
  - preconditions (Reminder 10), 2, 81–83, 211
  - project-linking software as, 14–15
  - purpose and writing style, 7–11
  - quality questions (Reminder 15), 11, 219
  - reference to another use case (underlined), 3, 113
  - scaling up, 143–144
  - situations for, 7–11
  - standards, 11, 132–137
  - subforms of, 7–11

Use cases (*cont.*)

- subfunctions (underwater fish/clam graphic, indigo/black, minus sign), 3, 7, 62–63, 66–67, 69, 142
  - “sufficient” use cases, 5
  - summary (cloud/kite graphic, white, plus sign), 3, 7, 62–67, 142, 144
  - system use cases (box graphic, grey/white), 3–7, 9–11, 41, 157–159, 216
  - techniques, 11
  - technology and data variations, 111–112
  - template for, 7
  - usage narratives, 17–19
  - user-goal level (sea-level waves graphic, blue, exclamation mark), 3–4, 6–7, 9–11
  - user goals stated with, 15–16
  - white-box use cases, 7, 40–41, 59–60, 218
- User goals
- context diagram in UML (Guideline 17), 243
  - use cases for stating, 15–16
  - use cases (sea-level waves graphic, blue, exclamation mark), 3–4, 6–7, 9–11, 62–64
- User interface (UI)
- design from use cases, 177–178
  - details, too many, 191–192, 194–202
- User stories, 187

## Validation

- checking versus (Guideline 7), 95
  - stakeholders' protection, 31
- Vision statements for scope, 51
- View of system's functions, 180–183
- “VW-Staging” (Cockburn), 142, 170
- Walters, Russell, 158–160, 194–202, 205
- Waves graphic, blue, exclamation mark (user-goal use cases), 3–4, 6–7, 9–11, 62–64
- White, cloud/kite graphic, plus sign (summary use cases), 3, 7, 62–67, 142, 144
- White-box use cases, 7, 40–41, 59–60, 218
- White/grey, box graphic (system use cases), 3–7, 9–11, 41, 157–159, 216
- White/grey, building graphic (business use cases), 3, 7, 41
- “Who has the ball?” (Guideline 2, Reminder 5), 90–91, 207
- Williams, Alan, 116–117
- Wirfs-Brock, Rebecca, 122, 235
- Word processors with hyperlinks for tool, 229
- Work breadth first (Reminder 17), 221–223
- Writing. *See also* Use cases
  - project plans, 180–186
  - styles, forces affecting, 128–132
- XP (eXtreme Programming), 187, 223–224
- Young, Steve, 38