

*The Addison-Wesley Signature Series*

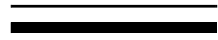
# USER STORIES APPLIED

FOR AGILE SOFTWARE  
DEVELOPMENT

MIKE COHN  
Foreword by Kent Beck

KENT BECK SIGNATURE  
BOOK





# User Stories Applied

---

# The Addison-Wesley Signature Series



**The Addison-Wesley Signature Series** provides readers with practical and authoritative information on the latest trends in modern technology for computer professionals. The series is based on one simple premise: great books come from great authors. Books in the series are personally chosen by expert advisors, world-class authors in their own right. These experts are proud to put their signatures on the covers, and their signatures ensure that these thought leaders have worked closely with authors to define topic coverage, book scope, critical content, and overall uniqueness. The expert signatures also symbolize a promise to our readers: you are reading a future classic.

## THE ADDISON-WESLEY SIGNATURE SERIES

SIGNERS: KENT BECK AND MARTIN FOWLER

**Kent Beck** has pioneered people-oriented technologies like JUnit, Extreme Programming, and patterns for software development. Kent is interested in helping teams do well by doing good — finding a style of software development that simultaneously satisfies economic, aesthetic, emotional, and practical constraints. His books focus on touching the lives of the creators and users of software.

**Martin Fowler** has been a pioneer of object technology in enterprise applications. His central concern is how to design software well. He focuses on getting to the heart of how to build enterprise software that will last well into the future. He is interested in looking behind the specifics of technologies to the patterns, practices, and principles that last for many years; these books should be usable a decade from now. Martin's criterion is that these are books he wished he could write.

---

### TITLES IN THE SERIES



*Implementation Patterns*

Kent Beck, ISBN 0321413091

*Test-Driven Development: By Example*

Kent Beck, ISBN 0321146530

*User Stories Applied: For Agile Software Development*

Mike Cohn, ISBN 0321205685

*Implementing Lean Software Development: From Concept to Cash*

Mary and Tom Poppendieck, ISBN 0321437381

---



*Refactoring Databases: Evolutionary Database Design*

Scott W. Ambler and Pramodkumar J. Sadalage, ISBN 0321293533

*Continuous Integration: Improving Software Quality and Reducing Risk*

Paul M. Duvall, with Steve Matyas and Andrew Glover, 0321336380

*Patterns of Enterprise Application Architecture*

Martin Fowler, ISBN 0321127420

*Refactoring HTML: Improving the Design of Existing Web Applications*

Elliotte Rusty Harold, ISBN 0321503635

*Beyond Software Architecture: Creating and Sustaining Winning Solutions*

Luke Hohmann, ISBN 0201775948

*Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*

Gregor Hohpe and Bobby Woolf, ISBN 0321200683

*Refactoring to Patterns*

Joshua Kerievsky, ISBN 0321213351

---

---

# User Stories Applied

*for Agile Software Development*

Mike Cohn

◆ 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 Addison-Wesley was aware of a trademark claim, the designations have been printed with 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.

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

U.S. Corporate and Government Sales  
(800) 382-3419  
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact:

International Sales  
(317) 581-3793  
international@pearsontechgroup.com

Visit Addison-Wesley on the Web: [www.awprofessional.com](http://www.awprofessional.com)

*Library of Congress Cataloging-in-Publication Data*

A catalog record for this book can be obtained from the Library of Congress

Copyright © 2004 by Pearson Education, Inc.

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

For information on obtaining permission for use of material from this work, please submit a written request to:

Pearson Education, Inc.  
Rights and Contracts Department  
75 Arlington Street, Suite 300  
Boston, MA 02116  
Fax: (617) 848-7047

ISBN 0-321-20568-5

Text printed in the United States on recycled paper at RR Donnelley Crawfordsville in Crawfordsville, Indiana.

13th Printing February 2009

*To Laura, for reading this one;  
To Savannah, for loving to read;  
To Delaney, for insisting you already know how to read.  
With readers like you, it's easy to write.*

*This page intentionally left blank*

# Contents

Foreword .....	xv
Acknowledgments .....	xvii
Introduction .....	xix
<b>PART I: Getting Started.....</b>	<b>1</b>
<b>Chapter 1: An Overview .....</b>	<b>3</b>
What Is a User Story?.....	4
Where Are the Details?.....	5
“How Long Does It Have to Be?” .....	7
The Customer Team.....	8
What Will the Process Be Like? .....	8
Planning Releases and Iterations.....	10
What Are Acceptance Tests? .....	12
Why Change?.....	13
Summary .....	15
Questions.....	15
<b>Chapter 2: Writing Stories .....</b>	<b>17</b>
Independent .....	17
Negotiable .....	18
Valuable to Purchasers or Users .....	20
Estimatable.....	22
Small .....	23
Testable .....	27
Summary .....	28
Developer Responsibilities .....	28
Customer Responsibilities .....	28
Questions.....	29



<b>Chapter 3:</b>	<b>User Role Modeling</b> . . . . .	<b>31</b>
	User Roles . . . . .	31
	Role Modeling Steps . . . . .	33
	Two Additional Techniques . . . . .	37
	What If I Have On-Site Users? . . . . .	39
	Summary . . . . .	40
	Developer Responsibilities . . . . .	40
	Customer Responsibilities . . . . .	41
	Questions . . . . .	41
<b>Chapter 4:</b>	<b>Gathering Stories</b> . . . . .	<b>43</b>
	Elicitation and Capture Should Be Illicit . . . . .	43
	A Little Is Enough, or Is It? . . . . .	44
	Techniques . . . . .	45
	User Interviews . . . . .	45
	Questionnaires . . . . .	47
	Observation . . . . .	48
	Story-Writing Workshops . . . . .	49
	Summary . . . . .	52
	Developer Responsibilities . . . . .	53
	Customer Responsibilities . . . . .	53
	Questions . . . . .	53
<b>Chapter 5:</b>	<b>Working with User Proxies</b> . . . . .	<b>55</b>
	The Users' Manager . . . . .	55
	A Development Manager . . . . .	57
	Salespersons . . . . .	57
	Domain Experts . . . . .	58
	The Marketing Group . . . . .	59
	Former Users . . . . .	59
	Customers . . . . .	59
	Trainers and Technical Support . . . . .	61
	Business or Systems Analysts . . . . .	61
	What to Do When Working with a User Proxy . . . . .	61
	Can You Do It Yourself? . . . . .	63
	Constituting the Customer Team . . . . .	63
	Summary . . . . .	64
	Developer Responsibilities . . . . .	65
	Customer Responsibilities . . . . .	65
	Questions . . . . .	66

<b>Chapter 6: Acceptance Testing User Stories</b> . . . . .	<b>67</b>
Write Tests Before Coding . . . . .	68
The Customer Specifies the Tests . . . . .	69
Testing Is Part of the Process. . . . .	69
How Many Tests Are Too Many?. . . . .	70
The Framework for Integrated Test. . . . .	70
Types of Testing . . . . .	72
Summary. . . . .	73
Developer Responsibilities. . . . .	73
Customer Responsibilities . . . . .	73
Questions . . . . .	74
<b>Chapter 7: Guidelines for Good Stories</b> . . . . .	<b>75</b>
Start with Goal Stories . . . . .	75
Slice the Cake . . . . .	75
Write Closed Stories . . . . .	76
Put Constraints on Cards . . . . .	77
Size the Story to the Horizon. . . . .	78
Keep the UI Out as Long as Possible. . . . .	79
Some Things Aren't Stories . . . . .	80
Include User Roles in the Stories . . . . .	80
Write for One User . . . . .	81
Write in Active Voice . . . . .	81
Customer Writes . . . . .	81
Don't Number Story Cards. . . . .	82
Don't Forget the Purpose . . . . .	82
Summary. . . . .	82
Questions . . . . .	83
<b>PART II: Estimating and Planning</b> . . . . .	<b>85</b>
<b>Chapter 8: Estimating User Stories</b> . . . . .	<b>87</b>
Story Points. . . . .	87
Estimate as a Team . . . . .	88
Estimating. . . . .	88
Triangulate . . . . .	90
Using Story Points. . . . .	91
What If We Pair Program? . . . . .	92
Some Reminders . . . . .	93
Summary. . . . .	94

	Developer Responsibilities . . . . .	94
	Customer Responsibilities . . . . .	95
	Questions . . . . .	95
<b>Chapter 9:</b>	<b>Planning a Release . . . . .</b>	<b>97</b>
	When Do We Want the Release? . . . . .	98
	What Would You Like in It? . . . . .	98
	Prioritizing the Stories . . . . .	99
	Mixed Priorities . . . . .	100
	Risky Stories . . . . .	101
	Prioritizing Infrastructural Needs . . . . .	101
	Selecting an Iteration Length . . . . .	103
	From Story Points to Expected Duration . . . . .	103
	The Initial Velocity . . . . .	104
	Creating the Release Plan . . . . .	105
	Summary . . . . .	106
	Developer Responsibilities . . . . .	106
	Customer Responsibilities . . . . .	107
	Questions . . . . .	107
<b>Chapter 10:</b>	<b>Planning an Iteration . . . . .</b>	<b>109</b>
	Iteration Planning Overview . . . . .	109
	Discussing the Stories . . . . .	110
	Disaggregating into Tasks . . . . .	111
	Accepting Responsibility . . . . .	113
	Estimate and Confirm . . . . .	113
	Summary . . . . .	115
	Developer Responsibilities . . . . .	115
	Customer Responsibilities . . . . .	116
	Questions . . . . .	116
<b>Chapter 11:</b>	<b>Measuring and Monitoring Velocity . . . . .</b>	<b>117</b>
	Measuring Velocity . . . . .	117
	Planned and Actual Velocity . . . . .	119
	Iteration Burndown Charts . . . . .	121
	Burndown Charts During an Iteration . . . . .	123
	Summary . . . . .	126
	Developer Responsibilities . . . . .	127
	Customer Responsibilities . . . . .	127
	Questions . . . . .	127

<b>PART III: Frequently Discussed Topics</b> . . . . .	<b>131</b>
<b>Chapter 12: What Stories Are Not</b> . . . . .	<b>133</b>
User Stories Aren't IEEE 830. . . . .	133
User Stories Are Not Use Cases. . . . .	137
User Stories Aren't Scenarios. . . . .	141
Summary. . . . .	143
Questions . . . . .	143
<b>Chapter 13: Why User Stories?</b> . . . . .	<b>145</b>
Verbal Communication. . . . .	145
User Stories Are Comprehensible . . . . .	148
User Stories Are the Right Size for Planning . . . . .	148
User Stories Work for Iterative Development . . . . .	149
Stories Encourage Deferring Detail . . . . .	150
Stories Support Opportunistic Development . . . . .	151
User Stories Encourage Participatory Design. . . . .	152
Stories Build Up Tacit Knowledge. . . . .	153
Why Not Stories? . . . . .	153
Summary. . . . .	154
Developer Responsibilities. . . . .	155
Customer Responsibilities . . . . .	155
Questions . . . . .	155
<b>Chapter 14: A Catalog of Story Smells</b> . . . . .	<b>157</b>
Stories Are Too Small . . . . .	157
Interdependent Stories. . . . .	157
Goldplating. . . . .	158
Too Many Details. . . . .	159
Including User Interface Detail Too Soon . . . . .	159
Thinking Too Far Ahead. . . . .	160
Splitting Too Many Stories . . . . .	160
Customer Has Trouble Prioritizing . . . . .	161
Customer Won't Write and Prioritize the Stories. . . . .	162
Summary. . . . .	162
Developer Responsibilities. . . . .	163
Customer Responsibilities . . . . .	163
Questions . . . . .	163
<b>Chapter 15: Using Stories with Scrum</b> . . . . .	<b>165</b>
Scrum Is Iterative and Incremental . . . . .	165

The Basics of Scrum . . . . .	166
The Scrum Team . . . . .	166
The Product Backlog . . . . .	167
The Sprint Planning Meeting . . . . .	168
The Sprint Review Meeting . . . . .	170
The Daily Scrum Meeting . . . . .	171
Adding Stories to Scrum . . . . .	173
A Case Study . . . . .	174
Summary . . . . .	175
Questions . . . . .	176
<b>Chapter 16: Additional Topics . . . . .</b>	<b>177</b>
Handling NonFunctional Requirements . . . . .	177
Paper or Software? . . . . .	179
User Stories and the User Interface . . . . .	181
Retaining the Stories . . . . .	184
Stories for Bugs . . . . .	185
Summary . . . . .	186
Developer Responsibilities . . . . .	186
Customer Responsibilities . . . . .	187
Questions . . . . .	187
<b>PART IV: An Example . . . . .</b>	<b>189</b>
<b>Chapter 17: The User Roles . . . . .</b>	<b>191</b>
The Project . . . . .	191
Identifying the Customer . . . . .	191
Identifying Some Initial Roles . . . . .	192
Consolidating and Narrowing . . . . .	193
Role Modeling . . . . .	195
Adding Personas . . . . .	197
<b>Chapter 18: The Stories . . . . .</b>	<b>199</b>
Stories for Teresa . . . . .	199
Stories for Captain Ron . . . . .	202
Stories for a Novice Sailor . . . . .	203
Stories for a Non-Sailing Gift Buyer . . . . .	204
Stories for a Report Viewer . . . . .	204
Some Administration Stories . . . . .	205
Wrapping Up . . . . .	206

<b>Chapter 19: Estimating the Stories</b> . . . . .	<b>209</b>
The First Story . . . . .	210
Advanced Search . . . . .	212
Rating and Reviewing . . . . .	213
Accounts . . . . .	214
Finishing the Estimates . . . . .	215
All the Estimates . . . . .	216
<b>Chapter 20: The Release Plan</b> . . . . .	<b>219</b>
Estimating Velocity . . . . .	219
Prioritizing the Stories . . . . .	220
The Finished Release Plan . . . . .	221
<b>Chapter 21: The Acceptance Tests</b> . . . . .	<b>223</b>
The Search Tests . . . . .	223
Shopping Cart Tests . . . . .	224
Buying Books . . . . .	225
User Accounts . . . . .	226
Administration . . . . .	227
Testing the Constraints . . . . .	228
A Final Story . . . . .	229
<b>PART V: Appendices</b> . . . . .	<b>231</b>
<b>Appendix A: An Overview of Extreme Programming</b> . . . . .	<b>233</b>
Roles . . . . .	233
The Twelve Practices . . . . .	234
XP's Values . . . . .	240
The Principles of XP . . . . .	241
Summary . . . . .	242
<b>Appendix B: Answers to Questions</b> . . . . .	<b>245</b>
Chapter 1, An Overview . . . . .	245
Chapter 2, Writing Stories . . . . .	246
Chapter 3, User Role Modeling . . . . .	247
Chapter 4, Gathering Stories . . . . .	248
Chapter 5, Working with User Proxies . . . . .	249
Chapter 6, Acceptance Testing User Stories . . . . .	249
Chapter 7, Guidelines for Good Stories . . . . .	249
Chapter 8, Estimating User Stories . . . . .	251
Chapter 9, Planning a Release . . . . .	251



Chapter 10, Planning an Iteration . . . . .	252
Chapter 11, Measuring and Monitoring Velocity . .	252
Chapter 12, What Stories Are Not . . . . .	253
Chapter 13, Why User Stories? . . . . .	254
Chapter 14, A Catalog of Story Smells . . . . .	255
Chapter 15, Using Stories with Scrum . . . . .	255
Chapter 16, Additional Topics . . . . .	256
<b>References . . . . .</b>	<b>259</b>
<b>Index . . . . .</b>	<b>263</b>

# Foreword

How do you decide what a software system is supposed to do? And then, how do you communicate that decision between the various people affected? This book takes on this complicated problem. The problem is difficult because each participant has different needs. Project managers want to track progress. Programmers want to implement the system. Product managers want flexibility. Testers want to measure. Users want a useful system. Creating productive conflict between these perspectives, coming to a single collective decision everyone can support, and maintaining that balance for months or years are all difficult problems.

The solution Mike Cohn explores in this book, *User Stories Applied*, is superficially the same as previous attempts to solve this problem—requirements, use cases, and scenarios. What’s so complicated? You write down what you want to do and then you do it. The proliferation of solutions suggests that the problem is not as simple as it appears. The variation comes down to what you write down and when.

User stories start the process by writing down just two pieces of information: each goal to be satisfied by the system and the rough cost of satisfying that goal. This takes just a few sentences and gives you information other approaches don't. Using the principle of the “last responsible moment,” the team defers writing down most details of the features until just before implementation.

This simple time shift has two major effects. First, the team can easily begin implementing the “juiciest” features early in the development cycle while the other features are still vague. The automated tests specifying the details of each feature ensure that early features continue to run as specified as you add new features. Second, putting a price on features early encourages prioritizing from the beginning instead of a panicked abortion of scope at the end in order to meet a delivery date.



Mike's experience with user stories makes this a book full of practical advice for making user stories work for your development team. I wish you clear, confident development.

Kent Beck  
Three Rivers Institute

# Acknowledgments

This book has benefitted greatly from comments from many reviewers. In particular I thank Marco Abis, Dave Astels, Steve Bannerman, Steve Berczuk, Lyn Bain, Dan Brown, Laura Cohn, Ron Crocker, Ward Cunningham, Rachel Davies, Robert Ellsworth, Doris Ford, John Gilman, Sven Gorts, Deb Hartmann, Chris Leslie, Chin Keong Ling, Philip, Keith Ray, Michele Sliger, Jeff Tatelman, Anko Tijman, Trond Wingård, Jason Yip, and a handful of anonymous reviewers.

My sincerest thanks to my formal reviewers: Ron Jeffries, Tom Poppendieck, and Bill Wake. Ron kept me honest and agile. Tom opened my eyes to many ideas I hadn't considered before. Bill kept me on track and shared with me his INVEST acronym. This book has been immeasurably improved by suggestions from each of these fine individuals with whom I am proud to have worked.

I also thank Lisa Crispin, author of *Testing Extreme Programming*, who encouraged me to write this book by telling me about her pleasant experience with Addison-Wesley. Without her encouragement, I never would have started.

Most of what I know I've argued about with Tod Golding over the last nine years. Tod and I agree more often than either of us knows, but I always learn something from our arguments. I am indebted to Tod for all he's taught me over the years. Much of this book has been greatly enriched because of my conversations with him.

Thanks to Alex Viggio and everyone at XP Denver where I was able to present an early version of many of the ideas in this book. Thank you, also, to Mark Mosholder and J. B. Rainsberger, who talked to me about how they use software instead of note cards. Thank you to Kenny Rubin, co-author of *Succeeding With Objects* with Adele Goldberg, whose obvious pride in their book helped me want to write again after a few years off.

A hearty thank you to Mark and Dan Gutrich, the founders of Fast401k, who have wholeheartedly embraced user stories and Scrum. Thank you as well to each of my coworkers at Fast401k, where we are well on our way to achieving our goal of being one of the best teams in Colorado.

There is no way to thank my family enough for all the time they did without me. Thank you to my wonderful daughters and princesses, Savannah and Delaney. A special thank you to my wonderful and beautiful wife, Laura, for doing so much even when I do so little.

I owe a huge debt of gratitude to the team at Addison-Wesley. Paul Petralia made the process enjoyable from start to finish. Michele Vincenti kept things moving. Lisa Iarkowski offered me invaluable FrameMaker help. Gail Cocker made my illustrations worth looking at. And Nick Radhuber brought it all together at the end.

And last, but far from least, thank you to Kent Beck for his wonderful insights, his time, and for including this book in his Signature Series.

# Introduction

I felt guilty throughout much of the mid-1990s. I was working for a company that was acquiring about one new company each year. Every time we'd buy a new company I would be assigned to run their software development group. And each of the acquired development groups came with glorious, beautiful, lengthy requirements documents. I inevitably felt guilty that my own groups were not producing such beautiful requirements specifications. Yet, my groups were consistently far more successful *at producing software* than were the groups we were acquiring.

I knew that what we were doing worked. Yet I had this nagging feeling that if we'd write big, lengthy requirements documents we could be even more successful. After all, that was what was being written in the books and articles I was reading at the time. If the successful software development teams were writing glorious requirements documents then it seemed like we should do the same. But, we never had the time. Our projects were always too important and were needed too soon for us to delay them at the start.

Because we never had the time to write a beautiful, lengthy requirements document, we settled on a way of working in which we would talk with our users. Rather than writing things down, passing them back and forth, and negotiating while the clock ran out, we talked. We'd draw screen samples on paper, sometimes we'd prototype, often we'd code a little and then show the intended users what we'd coded. At least once a month we'd grab a representative set of users and show them exactly what had been coded. By staying close to our users and by showing them progress in small pieces, we had found a way to be successful without the beautiful requirements documents.

Still, I felt guilty that we weren't doing things the way I thought we were supposed to.

In 1999 Kent Beck's revolutionary little book, *Extreme Programming Explained: Embrace Change*, was released. Overnight all of my guilt went away. Here was someone saying it was OK for developers and customers to talk rather than write, negotiate, and then write some more. Kent clarified a lot of

things and gave me many new ways of working. But, most importantly, he justified what I'd learned from my own experience.

Extensive upfront requirements gathering and documentation can kill a project in many ways. One of the most common is when the requirements document itself becomes a goal. A requirements document should be written only when it helps achieve the real goal of delivering some software.

A second way that extensive upfront requirements gathering and documentation can kill a project is through the inaccuracies of written language. I remember many years ago being told a story about a child at bath time. The child's father has filled the bath tub and is helping his child into the water. The young child, probably two or three years old, dips a toe in the water, quickly removes it, and tells her father "make it warmer." The father puts his hand into the water and is surprised to find that, rather than too cold, the water is already warmer than what his daughter is used to.

After thinking about his child's request for a moment, the father realizes they are miscommunicating and are using the same words to mean different things. The child's request to "make it warmer" is interpreted by any adult to be the same as "increase the temperature." To the child, however, "make it warmer" meant "make it closer to the temperature I call warm."

Words, especially when written, are a very thin medium through which to express requirements for something as complex as software. With their ability to be misinterpreted we need to replace written words with frequent conversations between developers, customers, and users. User stories provide us with a way of having just enough written down that we don't forget and that we can estimate and plan while also encouraging this time of communication.

By the time you've finished the first part of this book you will be ready to begin the shift away from rigorously writing down every last requirement detail. By the time you've finished the book you will know everything necessary to implement a story-driven process in your environment. This book is organized in four parts and two appendices.

- Part I: Getting Started—A description of everything you need to know to get started writing stories *today*. One of the goals of user stories is to get people talking rather than writing. It is the goal of Part I to get you talking as soon as possible. The first chapter provides an overview of what a user story is and how you'll use stories. The next chapters in Part I provide additional detail on writing user stories, gathering stories through user role modeling, writing stories when you don't have access to real end users, and testing user stories. Part I concludes with a chapter providing guidelines that will improve your user stories.

- **Part II: Estimating and Planning**—Equipped with a collection of user stories, one of the first things we often need to answer is “How long will it take to develop?” The chapters of Part II cover how to estimate stories in story points, how to plan a release over a three- to six-month time horizon, how to plan an ensuing iteration in more detail, and, finally, how to measure progress and assess whether the project is progressing as you’d like.
- **Part III: Frequently Discussed Topics**—Part III starts by describing how stories differ from requirements alternatives such as use cases, software requirements specifications, and interaction design scenarios. The next chapters in Part III look at the unique advantages of user stories, how to tell when something is going wrong, and how to adapt the agile process Scrum to use stories. The final chapter of Part III looks at a variety of smaller issues such as whether to write stories on paper note cards or in a software system and how to handle nonfunctional requirements.
- **Part IV: An Example**—An extended example intended to help bring everything together. If we’re to make the claim that developers can best understand user’s needs through stories then it is important to conclude this book with an extended story showing all aspects of user stories brought together in one example.
- **Part V: Appendices**—User stories originate in Extreme Programming. You do not need to be familiar with Extreme Programming in order to read this book. However, a brief introduction to Extreme Programming is provided in Appendix A. Appendix B contains answers to the questions that conclude the chapters.

*This page intentionally left blank*

## Chapter 3

---

# User Role Modeling

On many projects, stories are written as though there is only one type of user. All stories are written from the perspective of that user type. This simplification is a fallacy and can lead a team to miss stories for users who do not fit the general mold of the system's primary user type. The disciplines of usage-centered design (Constantine and Lockwood 1999) and interaction design (Cooper 1999) teach us the benefits of identifying user roles and personas prior to writing stories. In this chapter we will look at user roles, role modeling, user role maps, and personas and show how taking these initial steps leads to better stories and better software.

---

### User Roles<sup>1</sup>

Suppose we are building the BigMoneyJobs job posting and search site. This type of site will have many different types of users. When we talk about *user* stories, who is the user we're talking about? Are we talking about Ashish who has a job but always keeps an eye out for a better one? Are we talking about Laura, a new college graduate looking for her first professional job? Are we talking about Allan, who has decided he'll take any job that lets him move to Maui and windsurf every afternoon? Or are we talking about Scott, who doesn't hate his job but has realized it's time to move on? Perhaps we're talking about Kindra who was laid off six months ago and was looking for a great job but will now take anything in the northeastern United States.

Or should we think of the user as coming from one of the companies posting the jobs? Perhaps the user is Mario, who works in human resources and posts

---

1. Much of the discussion of user roles in this chapter is based on the work of Larry Constantine and Lucy Lockwood. Further information on user role modeling is available at their website at [www.foruse.com](http://www.foruse.com) or in *Software for Use* (1999).



new job openings. Perhaps the user is Delaney, who also works in human resources but is responsible for reviewing resumes. Or perhaps the user is Savannah, who works as an independent recruiter and is looking for both good jobs and good people.

Clearly we cannot write stories from a single perspective and have those stories reflect the experiences, backgrounds and goals of each of these users. Ashish, an accountant, may look at the site once a month just to keep his options open. Allan, a waiter, may want to create a filter to notify him any time any job on Maui gets posted but he won't be able to do that unless we make it easy. Kindra may spend hours each day looking for a job, broadening her search as time goes by. If Mario and Delaney work for a large company with many positions to fill, they may spend four or more hours a day on the site.

While each user comes to your software with a different background and with different goals, it is still possible to aggregate individual users and think of them in terms of *user roles*. A user role is a collection of defining attributes that characterize a population of users and their intended interactions with the system. So, we could look at the users in the preceding example and group them into roles as shown in Table 3.1 into roles this way.

**Table 3.1** *One possible list of roles for the BigMoneyJobs project.*

Role	Who
Job Seeker	Scott
First Timer	Laura
Layoff Victim	Kindra
Geographic Searcher	Allan
Monitor	Ashish
Job Poster	Mario, Savannah
Resume Reader	Delaney, Savannah

Naturally, there will be some overlap between different user roles. The Job Seeker, First Timer, Layoff Victim, Geographic Searcher, and Monitor roles will all use the job search features of the site. They may use them in different ways and at different frequencies, but much of how they use the system will be similar. The Resume Reader and Job Poster roles will probably overlap as well since these roles are both pursuing the same goal of finding good candidates.

Table 3.1 does not show the only possible way to group users of BigMoneyJobs into roles. For example, we could choose to include roles like Part-Timer,

Full-Timer and Contractor. In the rest of this chapter we'll look at how to come up with a list of roles and how to refine that list so that it is useful.

---

## Role Modeling Steps

We will use the following steps to identify and select a useful set of user roles:

- brainstorm an initial set of user roles
- organize the initial set
- consolidate roles
- refine the roles

Each of these steps is discussed in the following sections.

### Brainstorming an Initial Set of User Roles

To identify user roles, the customer and as many of the developers as possible meet in a room with either a large table or a wall to which they can tape or pin cards. It's always ideal to include the full team for the user role modeling that initiates a project but it's not necessary. As long as a reasonable representation of the developers is present along with the customer, you can have a successful session.

Each participant grabs a stack of note cards from a pile placed in the middle of the table. (Even if you plan to store the user roles electronically you should start by writing them on cards.) Start with everyone writing role names on cards and then placing them on a table, or taping or pinning them to a wall.

When a new role card is placed, the author says the name of the new role and nothing more. Since this is a brainstorming session, there is no discussion of the cards or evaluation of the roles. Rather, each person writes as many cards as he or she can think of. There are no turns, you don't go around the table asking for new roles. Each participant just writes a card whenever she thinks of a new role.

While brainstorming roles, the room will be filled with sounds of pens scratching on cards and will be punctuated by someone occasionally placing a new card and reading the name of the role. Continue until progress stalls and participants are having a hard time thinking up new roles. At that point you may not have identified all of the roles but you're close enough. Rarely does this need to last longer than fifteen minutes.

### A User Role Is One User

When brainstorming a project's roles, stick to identifying roles that represent a single user. For example, for the BigMoneyJobs project it may be tempting to write stories such as "A company can post a job opening." However, since a company as a whole cannot use the software, the story will be better if it refers to a role that represents an individual.

## Organizing the Initial Set

Once the group has finished identify roles, it's time to organize them. To do this, cards are moved around on the table or wall so that their positions indicate the relationships between the roles. Overlapping roles are placed so that their cards overlap. If the roles overlap a little, overlap the cards a little. If the roles overlap entirely, overlap the cards entirely. An example is shown in Figure 3.1.

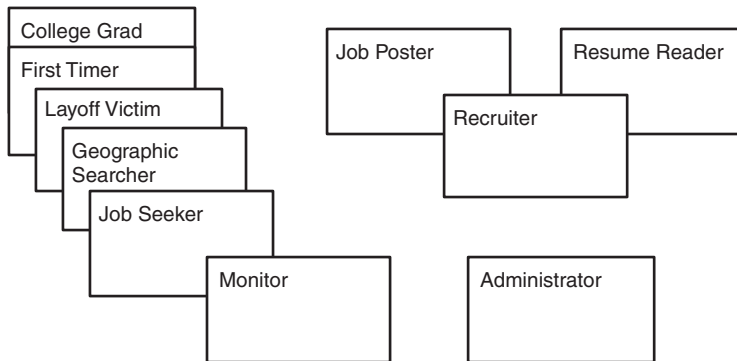


Figure 3.1 Organizing the user role cards on a table.

Figure 3.1 shows that the College Grad and First Timer, as those roles were intended by their card writers, overlap significantly. There's less but similar overlap among the other cards representing people who will use the site to search for jobs. The Monitor role card is shown with only a slight overlap because that role refers to someone who is relatively happy in her current job but likes to keep her eyes open.

To the right of the job-seeking roles in Figure 3.1 are the Job Poster, Recruiter, and Resume Reader role cards. The Recruiter role is shown overlapping both Job Poster and Resume Reader because a recruiter will both post ads

and read resumes. An Administrator role is shown also. This role represents users internal to BigMoneyJobs who will support the system.

---

### System Roles

As much as you can, stick with user roles that define people, as opposed to other systems. If you think it will help, then identify an occasional non-human user role. However, the purpose of identifying user roles is to make sure that we think really hard about the users that we absolutely, positively must satisfy with the new system. We don't need user roles for every conceivable user of the system, but we need roles for the ones who can make or break the success of the project. Since other systems are rarely purchasers of our system, they can rarely make or break the success of our system. Naturally, there can be exceptions to this and if you feel that adding a non-human user role helps you think about your system, then add it.

---

## Consolidating Roles

After the roles have been grouped, try to consolidate and condense the roles. Start with cards that are entirely overlapping. The authors of overlapping cards describe what they meant by those role names. After a brief discussion the group decides if the roles are equivalent. If equivalent, the roles can either be consolidated into a single role (perhaps taking its name from the two initial roles), or one of the initial role cards can be ripped up.

In Figure 3.1 the College Grad and First Timer roles are shown as heavily overlapping. The group decides to rip up the College Grad card since any stories for that user role would likely be identical to stories for a First Timer. Even though First Timer, Layoff Victim, Geographic Searcher and Job Seeker have significant overlap, the group decides that each represents a constituency that will be important to satisfy and the roles will have important but subtly different goals for how they use the BigMoneyJobs website.

When they look at the right side of Figure 3.1, the group decides that it is not worth distinguishing between a Job Poster and a Resume Reader. They decide that a Recruiter covers these two roles adequately and those cards are ripped up. However, the group decides that there are differences between an Internal Recruiter (working for a specific company) and an External Recruiter (matching candidates to jobs at any company). They create new cards for Internal Recruiter and External Recruiter, and consider these as specialized versions of the Recruiter role.

In addition to consolidating overlapping roles, the group should also rip up any role cards for roles that are unimportant to the success of the system. For example, the Monitor role card represents someone who is just keeping an eye on the job market. A Monitor may not switch jobs for three years. BigMoneyJobs can probably do quite well without paying attention to that user role. They decide they will be better off focusing on the roles that will be important to the success of the company, such as Job Seeker and the Recruiter roles.

After the team has consolidated the cards, they are arranged on the table or wall to show relationships between the roles. Figure 3.2 shows one of many possible layouts for the BigMoneyJobs role cards. Here a generic role, such as Job Seeker or Recruiter, is positioned above specialized versions of that role. Alternatively, cards can be stacked or positioned in any other way that the group desires in order to show whatever relationships they think are important.

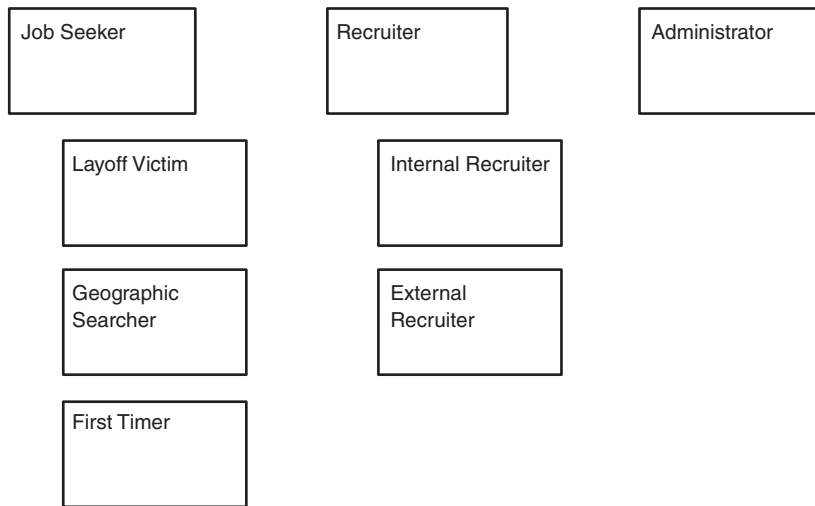


Figure 3.2 *The consolidated role cards.*

## Refining the Roles

Once we've consolidated roles and have a basic understanding for how the roles relate to each other, it is possible to model those roles by defining attributes of each role. A role attribute is a fact or bit of useful information about the users who fulfill the role. Any information about the user roles that distinguishes one role from another may be used as a role attribute. Here are some attributes worth considering when preparing any role model:

- The frequency with which the user will use the software.
- The user's level of expertise with the domain.
- The user's general level of proficiency with computers and software.
- The user's level of proficiency with the software being developed.
- The user's general goal for using the software. Some users are after convenience, others favor a rich experience, and so on.

Beyond these standard attributes you should consider the software being built and see if there are any attributes that might be useful in describing its users. For instance, for the BigMoneyJobs website you may want to consider whether the user role will be looking for a part-time or full-time job.

As you identify interesting attributes for a role, write notes on the role card. When finished, you can hang the role cards in a common area used by the team so they can be used as reminders. A sample user role card is shown in Figure 3.3.

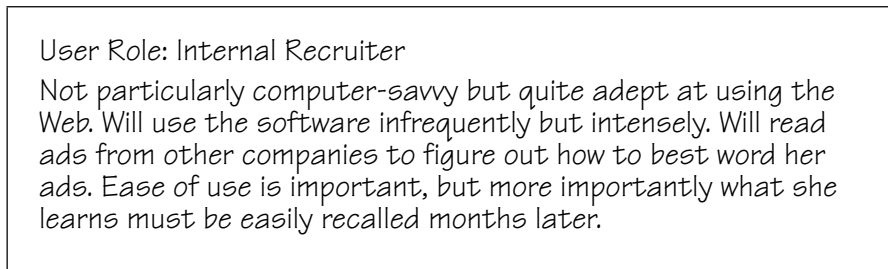


Figure 3.3 *A sample user role card.*

---

## Two Additional Techniques

We could stop right now if we want to. By now a team might have spent an hour—almost certainly no more than that—and they will have put more thought into the users of their software than probably 99% of all software teams. Most teams should, in fact, stop at this point. However, there are two additional techniques that are worth pointing out because they may be helpful in thinking about users on some projects. Only use these techniques if you can anticipate a direct, tangible benefit to the project.

## Personas

Identifying user roles is a great leap forward, but for some of the more important user roles, it might be worth going one step further and creating a *persona* for the role. A persona is an imaginary representation of a user role. Earlier in this chapter we met Mario who is responsible for posting new job openings for his company. Creating a persona requires more than just adding a name to a user role. A persona should be described sufficiently that everyone on the team feels like they know the persona. For example, Mario may be described as follows:

Mario works as a recruiter in the Personnel department of SpeedyNetworks, a manufacturer of high-end networking components. He's worked for SpeedyNetworks six years. Mario has a flex-time arrangement and works from home every Friday. Mario is very strong with computers and considers himself a power user of just about all the products he uses. Mario's wife, Kim, is finishing her Ph.D. in chemistry at Stanford University. Because SpeedyNetworks has been growing almost consistently, Mario is always looking for good engineers.

If you choose to create personas for your project, be careful that enough market and demographic research has been done that the personas chosen truly represent the product's target audience.

This persona description gives us a good introduction to Mario. However, nothing speaks as loudly as a picture, so you should also find a picture of Mario and include that with the persona definition. You can get photographs all over the web or you can cut one from a magazine. A solid persona definition combined with a photograph will give everyone on the team a thorough introduction to the persona.

Most persona definitions are too long to fit on a note card, so I suggest you write them on a piece of paper and hang them in the team's common space. You do not need to write persona definitions for every user role. You may, however, think about writing a persona definition for one or two of the primary user roles. If the system you are building is such that it is absolutely vital that the product satisfy one or two user roles, then those user roles are candidates for expansion into personas.

Stories become much more expressive when put in terms of a user role or persona. After you have identified user roles and possibly a persona or two, you can begin to speak in terms of roles and personas instead of the more generic "the user." Rather than writing stories like "A user can restrict job searches to specific geographic regions" you can write "A Geographic Searcher can restrict

his job searches to a specific geographic region.” Hopefully writing a story this way reminds the team about Allan who is looking for any job on Maui. Writing some stories with user role or persona names does not mean that other roles cannot perform those stories; rather, it means that there is some benefit in thinking about a specific user role or persona when discussing or coding the story.

## Extreme Characters

Djajadiningrat and co-authors (2000) have proposed a second technique you might want to think about: the use of extreme characters when considering the design of a new system. They describe an example of designing a Personal Digital Assistant (PDA) handheld computer. They advise that instead of designing solely for a typical sharp-dressed, BMW-driving management consultant, the system designers should consider users with exaggerated personalities. Specifically, the authors suggest designing the PDA for a drug dealer, the Pope, and a twenty-year-old woman who is juggling multiple boyfriends.

It is very possible that considering extreme characters will lead you to stories you would be likely to miss otherwise. For example, it is easy to imagine that the drug dealer and a woman with several boyfriends may each want to maintain multiple separate schedules in case the PDA is seen by the police or a boyfriend. The Pope probably has less need for secrecy but may want a larger font size.

So, while extreme characters may lead to new stories, it is hard to know whether those stories will be ones that should be included in the product. It is probably not worth much investment in time, but you might want to experiment with extreme characters. At a minimum, you can have a few minutes of fun thinking about how the Pope might use your software and it just may lead to an insight or two.

---

## What If I Have On-Site Users?

The user role modeling techniques described in this chapter are still useful even if you have real, live users in your building. Working with real users will strongly improve your likelihood of delivering the desired software. However, even with real users there is no guarantee that you have the right users or the right mix of users.



To decrease the likelihood of failing to satisfy important users, you should do some simple role modeling on projects even when you have available internal users.

---

## Summary

- Most project teams consider only a single type of user. This leads to software that ignores the needs of at least some user types.
- To avoid writing all stories from the perspective of a single user, identify the different user roles who will interact with the software.
- By defining relevant attributes for each user role, you can better see the differences between roles.
- Some user roles benefit from being described by personas. A persona is an imaginary representation of a user role. The persona is given a name, a face, and enough relevant details to make them seem real to the project members.
- For some applications, extreme characters may be helpful in looking for stories that would otherwise be missed.

---

## Developer Responsibilities

- You are responsible for participating in the process of identifying user roles and personas.
- You are responsible for understanding each of the user roles or personas and how they differ.
- While developing the software, you are responsible for thinking about how different user roles may prefer the software to behave.
- You are responsible for making sure that identifying and describing user roles does not go beyond its role as a tool in the process.

---

## Customer Responsibilities

- You are responsible for looking broadly across the space of possible users and identifying appropriate user roles.
- You are responsible for participating in the process of identifying user roles and personas.
- You are responsible for ensuring that the software does not focus inappropriately on a subset of users.
- When writing stories you will be responsible for ensuring that each story can be associated with at least one user role or persona.
- While developing the software, you are responsible for thinking about how different user roles may prefer the software to behave.
- You are responsible for making sure that identifying and describing user roles does not go beyond its role as a tool in the process.

---

## Questions

- 3.1 Take a look at the eBay website. What user roles can you identify?
- 3.2 Consolidate the roles you came up with in the previous question and show how you would lay out the role cards. Explain your answer.
- 3.3 Write persona descriptions for the one most important user role.

*This page intentionally left blank*

# Index

- A**  
Acceptance testing, 12–13, 15, 67–74  
    customer responsibilities, 73  
    customer specification of the tests, 69  
    developer responsibilities, 73  
    Framework for Integrated Test (FIT), 70–71  
    South Coast Nautical Supplies (example project), 223–230  
    administration, 227–228  
    buying books, 225–226  
    constraint testing, 228–229  
    final story, 229–230  
    search tests, 223–224  
    shopping cart tests, 224–225  
    user accounts, 226–227  
    testing as part of the process, 69–70  
    types of testing, 72  
    writing tests, 12–13  
        before coding, 68–69  
        writing too many tests, 70  
Accounts, South Coast Nautical Supplies (example project), 214–215  
Active voice, writing in, 81  
Administration stories, South Coast Nautical Supplies (example project), 205–206  
*Agile Development with Scrum* (Schwaber/Beedle), 165  
Agile processes, 111  
Agile usage-centered design, 182  
Anderson, Ann, 233  
Astels, Dave, 237
- B**  
Bad smells:  
    customer responsibilities, 163  
    customer won't write and prioritize the stories, 162  
    developer responsibilities, 163  
    goldplating, 158–159  
    including user interface detail too soon, 159–160  
    interdependent stories, 157–158  
    prioritizing difficulties, 161–162  
    small stories, 157  
    splitting too many stories, 160–161  
    thinking too far ahead, 160  
    too many details, 159  
Beck, Kent, 97, 136, 233–234, 236, 239  
Beedle, Mike, 165, 171  
Berczuk, Steve, 136  
BigMoneyJobs example story, 4–6, 23  
Black, J. B., 148  
Boehm, Barry, 88, 101  
Bower, G. H., 148  
Brainstorming:  
    initial set of user roles, 33–34  
    with low-fidelity prototyping, 49
- C**  
Capture, 43  
Card, 4, See also Note cards; Story cards  
    annotations on, 6–7  
    writing on the back of, 7  
Carroll, John M., 135, 141–142  
Central Limit Theorem, 92  
Certification, 180  
Changing priorities, 110  
Clements, Paul C., 151  
ClickTactics, choosing software at, 180  
Closed stories, 76–77  
Cockburn, Alistair, 137, 140, 150, 237  
Cohn, Mike, 173  
Collocated team, 105  
Colors, 185  
Combined stories, 18, 26–27  
Competitive analysis, 65  
Complex stories, 24–26  
    splitting, 25  
Compound stories, 24–25  
    disaggregating, 25  
Confirmation, 4  
Connection pools, 5  
Consistent iterations, 103  
Constantine, Larry, 31, 59, 140, 148, 181–182  
Constraints, examples of, 77–78  
Context-free questions, 46–47  
Cooper, Alan, 135  
Corporate desktop productivity software, 59  
Cost, and priority, 100

- Could-have features, release plan, 98–99
- Credit card validity, 71
- Cunningham, Ward, 70, 183
- Customer responsibilities:
  - bad smells, 163
  - gathering stories, 53
  - iteration planning, 116
  - release planning, 107
  - user proxies, 65
  - user role remodeling, 41
  - user stories, estimating, 95
  - velocity, measuring/monitoring, 127
  - writing stories, 28
- Customer team, 8, 15
  - constituting, 63–64
  - story writing by, 9
- Customers, as user proxies, 65
- D**
- Daily burndown chart, 123–126
- Daily scrum, 166, 171–172
- Davies, Rachel, 4, 140
- Decision-making, basis of, 4
- Defect trackers, 179
- Deferring detail, 14, 150
- Dependencies between stories, 17–18
- Details, 5–7
  - development team–customer discussions about, 6–7
  - as tests, 20
  - too many details, as bad smell, 159
  - user stories with too much detail, 19–20
- Developer responsibilities:
  - acceptance testing user stories, 73
  - bad smells, 163
  - gathering stories, 53
  - iteration planning, 115
  - note cards, 186, 187
  - release planning, 106
  - software system, 186, 187
  - user proxies, 65
  - user role remodeling, 40
  - user stories, 155
  - user stories, estimating, 94
  - velocity, measuring/monitoring, 127
  - writing stories, 28
- Development managers, 57, 64
- Diaspar Software Services, 181
- Disaggregation, 25, 94, 111–112
  - guidelines, 112
- Domain experts, 58–59, 65
- Domain knowledge, lack of, 23
- DSDM, and MoSCoW rules, 98
- DSDM: Business Focused Development* (Stapleton), 98
- E**
- Early releases, 63, 65
- Elicitation, 43
- Empirical design, 152
- Epics, 6, 94
  - categories of, 24
- Essential use cases, 140
- Estimatable stories, 22–23
- Estimating:
  - approach, 88–90
  - as a team, 88
  - triangulating an estimate, 90–91
- Example project, See South Coast Nautical Supplies (example project):
- Example user story, 4–5
- Expectation maximization, 101
- Extra detail, 19
- Extreme characters, 39
- Extreme Programming Explained: Embrace Change* (Beck), 233
- Extreme Programming Explored* (Wake), 17, 233
- Extreme Programming Installed* (Jeffries, Anderson, and Hendrickson), 233
- Extreme Programming (XP), 8, 22, 88, 179, 233–243
  - coach, 234
  - customer role, 233
  - principles of, 241–242
  - programmer role, 234
  - roles, 233–234
  - teams, 26
  - twelve XP practices, 234–240
  - coding standards, 239
  - continuous integration, 240
  - metaphor, 239
  - on-site customer, 240
  - pair programming, 237–238
  - Planning Game, 235–236
  - refactoring, 236
  - simple design, 239
  - small releases, 235
  - sustainable pace, 238
  - team code ownership, 238–239
  - testing, 236–237
  - values, 240–241
- F**
- Factors critical to project success, determining, 64
- First iteration, 9
- FitNesse, 71, 181
- Fowler, Martin, 150, 236
- Framework for Integrated Test (FIT), 70–71
- G**
- Gathering stories, 43–52
  - customer responsibilities, 53
  - developer responsibilities, 53
  - elicitation and capture, 43–44
  - observation, 48–49
  - questionnaires, 47–48
  - story-writing workshops, 49–52
  - techniques, 45
  - user interviews, 45–47
  - write stories at different levels of detail, 44–45
- Gilb, Tom, 101
- Goal stories, starting with, 75
- Goldplating, 158–159
  - example of, 158
- Good stories:
  - attributes of, 17
  - guidelines for, 75–83
    - active voice, writing in, 81
    - closed stories, 76–77
    - customer as writer of the stories, 81–82
    - including user roles in the stories, 80–81
    - putting constraints on cards, 77–78
    - sizing the story, 78–79

- story cards, numbering, 82
    - user interface, 79–80
    - using other formats, 80
    - writing for one user, 81
  - guidelines for writing:
    - goal stories, starting with, 76
    - splitting stories
  - Grenning, James, 137
  - Guindon, Raymonde, 151
- H**
- Hendrickson, Chet, 233
  - Highly dependent stories, 17–18
  - High-priority stories, 183
  - Historical values, and initial velocity, 104
- I**
- IEEE 830:
    - compared to user stories, 133–136
    - requirements, 135–136
  - Incremental process:
    - defined, 166
    - Scrum as, 166
  - Independent stories, 17–18
  - Individual ownership, 238–239
  - Infrastructural needs, and prioritization of needs, 101–103
  - Initial collection of stories, 209
  - Initial set of user roles:
    - brainstorming, 33–34
    - organizing, 34–35
  - Initial stories, 9
  - Initial velocity, 104–105
  - Inmates Are Running the Asylum, The* (Cooper), 135
  - Institute of Electrical and Electronics Engineers (IEEE), 133
  - Interdependent stories, bad smells, 157–158
  - ISO (International Organization for Standardization), 180
  - Iteration burndown charts, 121–123
    - chart size, 126
    - daily burndown chart, 123–125
    - usefulness of, 122
  - Iteration length, selecting, 103
  - Iteration planning, 10–12, 109–116
    - accepting responsibility, 113
    - customer responsibilities, 116
    - developer responsibilities, 115
    - disaggregating into tasks, 111–112
    - discussing the stories, 110
    - estimation and confirmation, 113–114
    - general sequence of activities for iterative planning meeting, 109
  - Iterations, 9–10
    - length, 9, 103
    - planning, 10–12, 109–116
  - Iterative development, and user stories, 149–150
  - Iterative process, 14
    - defined, 165
    - Scrum as, 165–166
- J**
- Jacobson, Ivar, 137
  - Jeffries, Ron, 4, 233
  - Joint Application Design (JAD) sessions, 49
- K**
- Kerievsky, Joshua, 87
  - Kuhn, Sarah, 152
- L**
- Large stories, temporarily skipping, 11–12
  - Lockwood, Lucy, 31, 59, 140, 148, 182
  - Low-fidelity prototype, 49–50
    - throwing away, 51
- M**
- Mad About You*, 89
  - Marketing group, 64
  - Martin, Bob, 71, 77
  - Martin, Micah, 71
  - Mixed priorities, 100–101
  - MoSCoW rules, 98
  - Muller, Michael J., 152
  - Multiple user proxies, using, 65
  - Must-have features, release plan, 98
- N**
- Namioka, Aki, 152
  - Negotiable stories, 18–19
  - Newkirk, James, 77
  - Non-functional requirements:
    - accuracy, 178
    - capacity, 179
    - as constraints on system behavior, 177–179
    - handling, 177–179
    - interoperability, 179
    - maintainability, 179
    - performance, 178
    - portability, 179
    - reusability, 178
    - types of, 177–178
  - Note cards, 4
    - colors of, 185
    - limitations of, 179–180
  - Note, story card with, 7
- O**
- Observation, 48–49
  - On-site users, 39–40
  - Open-ended questions, 46–47
- P**
- Pair programming, 92–93, 237–238
  - Parnas, David L., 151
  - Participatory design, 152
  - Performance testing, 72
  - Personal digital assistant (PDA) handheld computer, 39
  - Personas, 38–39
    - South Coast Nautical Supplies (example project), 197–198
  - Planned velocity, 119–120
  - Planning releases, 97–107
    - and iterations, 10–12
  - Poppendieck, Tom, 147, 159
  - Positioning user role cards, 193
  - Potentially shippable product increment, 170
  - Precision, and story size, 93
  - Prescriptive processes, 44
  - Priorities, changing, 110
  - Prioritizing stories, 98–101
    - customer has trouble prioritizing, 161–162
    - high-level stories, 183
    - infrastructural needs, 101–103
    - initial velocity, 104–105

- iteration length, selecting, 103
  - mixed priorities, 100–101
  - risky stories, 101
  - from story points to expected duration, 103
  - Process, 8–10
  - Product backlog, 166, 167–168
    - sample list, 168
    - and user stories, 173
  - Product development roadmap, 97
  - Product owner, 167
  - Project champion, identifying, 64
  - Project failures, 3
  - Purchaser, distinction between user and, 20–21
- Q**
- Questionnaires, 47–48
- R**
- Rainsberger, J. B., 181
  - Refactoring Workbook* (Wake), 17
  - Release plan, 9–12, 15
    - could-have features, 98–99
    - creating, 105–106
    - customer responsibilities, 107
    - developer responsibilities, 106
    - initiating, 97
    - MoSCoW rules, 98
    - must-have features, 98
    - prioritizing stories, 10, 98–101
    - putting too much faith in, 106
    - release date, 98
    - should-have features, 98–99
    - South Coast Nautical Supplies (example project), 219–222
    - finished plan, 221–222
    - won't have features, 98–99
  - Releases, planning, 9–10
  - Reminders, story cards as, 18–20
  - “Requirements engineering” efforts, 160
  - Retaining user stories, 184–185
  - Risky stories, 101
  - Role attribute, 36–37
  - Role cards, 33
  - Role modeling steps, 33–37
- S**
- Salespersons, 64
  - Scenarios, 137
    - compared to user stories, 141–142
  - Schuler, Douglas, 152
  - Schwaber, Ken, 165–166
  - Scrum, 9
    - adding stories to, 173–174
    - basics of, 166
    - as both iterative and incremental process, 165–166
    - case study, 174–175
    - main rules of, 169
    - Scrum team, 166–167
    - sprint planning meeting, 168–169
    - sprint review meeting, 169, 170–171
    - using stories with, 165–176
  - ScrumMaster, 167–168, 170–172
  - Select Scope Manager, 179
  - Short iterations, 103
  - Should-have features, release plan, 98–99
  - Small stories, bad smells, 157
  - Software Configuration Management Patterns* (Berczuk), 136
  - Software development
    - projects, predicting, 3
  - Software for Use* (Constantine and Lockwood), 311
  - Software requirements, as a communication problem, 3
  - Software system, developer responsibilities, 186, 187
  - Sorting stories, 99
  - South Coast Nautical Supplies (example project):
    - acceptance tests, 223–230
    - administration, 227–228
    - buying books, 225–226
    - constraint testing, 228–229
    - search tests, 223–224
    - shopping cart tests, 224–225
    - user accounts, 226–227
  - accounts, 214–215
  - administration stories, 205–206
  - advanced search, 212–213
  - complete list of stories and estimates, 216–217
  - consolidating and narrowing, 193–195
  - defined, 191
  - estimating the stories, 209–217
  - finishing the estimates, 215
  - identifying initial roles, 192
  - identifying the customer, 191–192
  - personas, adding, 197–198
  - prioritizing the stories, 220–221
  - rating and reviewing, 213–214
  - release plan, 219–222
    - finished, 221–222
  - stories for a Non-Sailing Gift Buyer, 204
  - stories for a novice sailor, 203
  - stories for a Report Viewer, 204–205
  - stories for Captain Ron, 202–203
  - stories for Teresa, 199–202
  - user role modeling, 195–197
  - user roles, 191–198
  - velocity, estimating, 219
- Spike, 22
  - putting in a different iteration, 26
- Spiral model (Boehm), 101
- Splitting stories, 18, 23, 24–26, 75–76
  - splitting too many stories, as bad smell, 160–161
- Sprint backlog, 166–167, 170
- Sprint goal, 168
- Sprint planning meeting, 168–170
  - user stories in, 173–174
- Sprint review meeting, 169, 170–171, 174
  - user stories in, 174
- Sprints, 166
- Stapleton, Jennifer, 98
- Story cards:
  - defined, 18
  - limitations of, 179–180
  - main purpose of, 82
  - numbering, 82
  - as reminders, 18–20, 82

- writing notes on the back of, 67
    - writing on the back of, 7
    - writing test on the back of, 13
  - Story cost, 10–11
  - Story points, 87–88, 91–92, 94
  - Story size, 23–27
    - and precision, 93
  - Story smells, see Bad smells:
  - Story writing process, 8–10
    - initial stories, 9
  - Story-driven projects, process, 8–10
  - Story-writing workshops, 49–52
    - contributors of, 52
    - focus of, 51–52
  - Stress testing, 72
  - Surveys, 46
  - System roles, 35
- T**
- Technical support personnel, 61
  - Test descriptions, 7–8
  - Testable stories, 27
  - Test-driven development, 237
  - Testing:
    - acceptance, 12–13, 15, 67–74
    - for bugs, 72
    - Framework for Integrated Test (FIT), 70–71
    - performance, 72
    - stress, 72
    - as a two-step process, 67
    - usability, 72
    - user interface, 72
  - Tests, writing before coding, 68–69
  - Themes, 97
  - Timebox, 22
  - Time-constrained projects, and deferred detail, 150
  - Trainers, 61
  - Trainers and technical support personnel, as user proxies, 65
  - Trawling for requirements, 43–44
  - Turner, T. J., 148
  - Twelve XP practices, 234–240
- U**
- Unified Process, 8, 137
  - Untestable stories, 27
  - Usability testing, 72
  - Use case briefs, 140
  - User, distinction between purchaser and, 20–21
  - User interface guidelines, 80
  - User interface testing, 72
  - User interface (UI):
    - details, including too soon, 159–160
    - writing two, 183
  - User Interviews, 45–47
    - open-ended and context-free questions, 46–47
  - User proxies, 55–66
    - customer responsibilities, 65
    - customers, 59–60
    - developer responsibilities, 65
    - development manager, 57
    - domain experts, 58–59
    - former users, 59
    - marketing group, 59
    - salespersons, 57–58
    - trainers and technical support personnel, 61
    - users' manager, 55–56
    - what to do when working with, 61–63
  - User role cards, 33–36, 192
    - positioning, 193
    - sample, 37
  - User role modeling, 9, 31–41
    - consolidating roles, 35–36
    - customer responsibilities, 41
    - developer responsibilities, 40
    - extreme characters, 39
    - initial set of user roles:
      - brainstorming, 33–34
      - organizing, 34–35
      - on-site users, 39–40
      - personas, 38–39
      - refining roles, 36–37
      - role attributes, 36
      - South Coast Nautical Supplies (example project), 195–197
      - steps in, 33–37
      - user roles, 31–33
  - User roles:
    - attributes, 36–37
    - brainstorming an initial set of, 33–34
    - consolidating, 35–36
    - identifying, 33
    - identifying roles that represent a single user, 34
    - including in user stories, 80–81
    - initial set of:
      - brainstorming, 33–34
      - organizing, 34–35
      - refining, 36–37
    - role attribute, 36–37
    - South Coast Nautical Supplies (example project), 191–198
  - User stories:
    - advantages over alternative approaches, 13–14
    - aspects of, 4
    - augmenting in requirements documentation style, 6
    - comprehensibility of, 148
    - customer responsibilities, 155
    - and the daily Scrum meeting, 174
    - and deferring detail, 150
    - defined, 4–5
    - descriptions, 4
    - developer responsibilities, 155
    - estimating, 87–95
      - approach, 88–90
      - customer responsibilities, 95
      - developer responsibilities, 94
      - pair programming, 92–93
      - story points, 87–88, 91–92
      - as a team, 88
      - triangling an estimate, 90–91
    - IEEE 830 compared to, 133–136
    - and iterative development, 149–150
    - and participatory design, 152
    - prioritizing, 98–101
    - and the product backlog, 173
    - relationship between bug reports and, 185
    - representing functionality valued by users, 5
    - retaining, 184–185
    - scenarios compared to, 141–142



- size for planning, 148–149
  - size of, 6
  - splitting, 6, 12
  - in the sprint planning meeting, 173–174
  - in the sprint review meeting, 174
  - support for opportunistic development, 151–152
  - tacit knowledge, build up of, 153
  - and technical jargon, 14
  - test descriptions, 7–8
  - with too much detail, 19
  - use cases compared to, 137–141
  - and the user interface (UI), 26, 79–80, 139–140, 159–160, 181–183
  - and verbal communication, 145–148
  - User task forces, 62, 65
  - Users' manager, 64
  - Users stories, initial collection of, 209–210
- V**
- Value to purchasers/users, 20–22
  - Velocity, 9–10, 15, 91–92, 113
    - actual, 119–120
    - calculations, 119
    - guessing at, 104–105
    - initial, 104–105
    - iteration burndown charts, 121–123
    - measuring, 117–119
    - measuring/monitoring:
      - customer responsibilities, 127
      - developer responsibilities, 127
    - planned, 119–120
  - Verbal communication, 145–148
  - VersionOne, 179
- W**
- Wake, Bill, 17, 76, 233
  - Waterfall-oriented process, 8
  - “White book” (Beck), 234
  - Wideband Delphi approach, 88
  - Wikis, 179
  - Williams, Laurie, 237
  - Won't-have features, release plan, 98–99
  - Writing stories, 17–29
    - combined stories, 18
    - combining stories, 26–27
    - customer responsibilities, 28
    - developer responsibilities, 28
    - estimatable stories, 22–23
    - highly dependent stories, 17–18
    - independent stories, 17–18
    - negotiable stories, 18–19
    - small stories, 23–27
    - splitting stories, 18
    - story size, 23–27
    - testable stories, 27
- X**
- XP, See Extreme Programming (XP)
  - XPlanner, 179