The Addison-Wesley Signature Series

A Kent Beck Signature Book

User Stories Applied

For Agile Software Development

Mike Cohn

Foreword by Kent Beck

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

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

Library of Congress Cataloging-in-Publication Data

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

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

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, Phlip, 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 writes 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[1]

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 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 dicsussion 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.

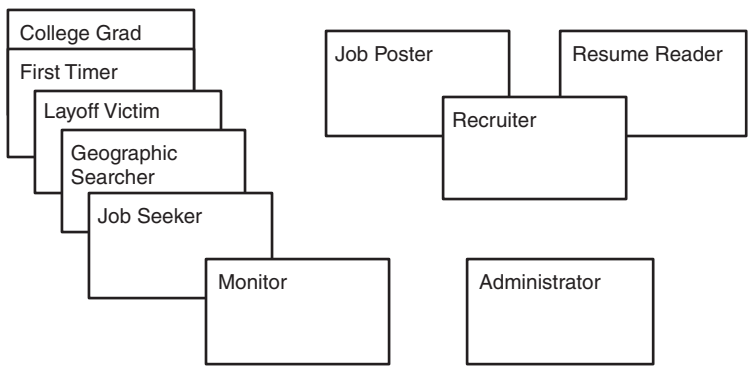

Figure 3.1 *Organizing the user role cards on a table.*

Figure 3.1 shows that the College Grad and First Timer, as those roles were intended by their card writers, overlap signficantly. 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 discusson 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. BigMoney-Jobs 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.

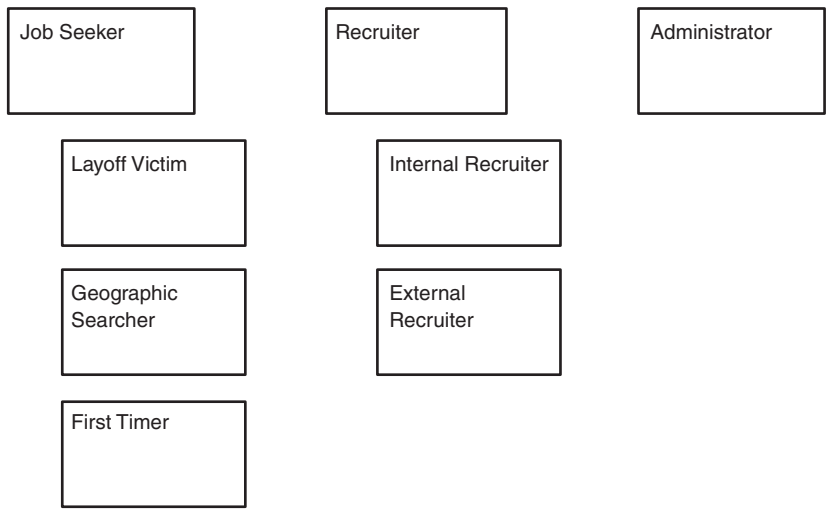

Figure 3.2 *The consolidated role cards.*

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.

User Role: Internal Recruiter

Not particularly computer-savvy but quite adept at using the Web. Will use the software infrequently but intensely. Will read ads from other companies to figure out how to best word her ads. Ease of use is important, but more importantly what she learns must be easily recalled months later.

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. Spefically, 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