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.