Agile Software Requirements

Lean Requirements Practices for Teams, Programs, and the Enterprise

Dean Leffingwell

Foreword by **Don Reinertsen**

Agile Software Development Series

Alistair Cockburn and Jim Highsmith, Series Editors

Praise for *Agile Software Requirements*

"In my opinion, there is no book out there that more artfully addresses the specific needs of agile teams, programs, and portfolios all in one. I believe this book is an organizational necessity for any enterprise."

—Sarah Edrie, Director of Quality Engineering, Harvard Business School

"*Agile Software Requirements* and Mr. Leffingwell's teachings have been very influential and inspiring to our organization. They have allowed us to make critical cultural changes to the way we approach software development by following the framework he's outlined here. It has been an extraordinary experience."

—Chris Chapman, Software Development Manager, Discount Tire

"This book supplies empirical wisdom connected with strong and very well-structured theory of succeeding with software projects of different scales. People new to agile, practitioners, or accomplished agilists—we all were waiting for such a book."

—Oleksandr (Alex) Yakyma, Agile Consultant, www.enter-Agile.com

"This book presents practical and proven agile approaches for managing software requirements for a team, collaborating teams of teams, and all across the enterprise. However, this is not *only* a great book on agile requirements engineering; rather, Leffingwell describes the bigger picture of how the enterprise can achieve the benefits of business agility by implementing lean product development flow. His 'Big Picture' of agile requirements is an excellent reference for any organization pursuing an intrinsically lean software development operational mode. Best of all, we've applied many of these principles and practices at Nokia (and even helped create some of them), and therefore we know they *work*.

—Juha-Markus Aalto, Agile Change Program Manager, Nokia Corporation

"This pragmatic, easy-to-understand, yet thought-provoking book provides a hands-on guide to addressing a key problem that enterprises face: How to make requirements practices work effectively in large-scale agile environments. Dean Leffingwell's focus on lean principles is refreshing and much needed!"

—Per Kroll, author, and Chief Architect for Measured Improvements, IBM

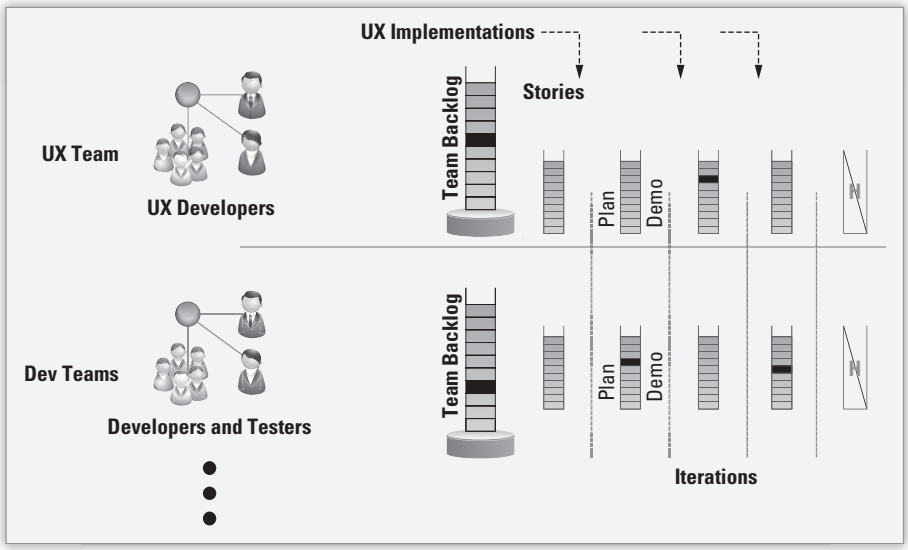

Figure 7–3 Centralized UX development

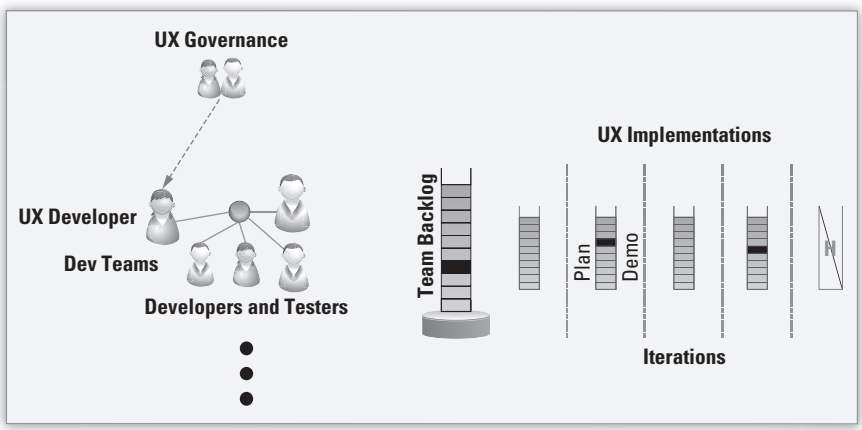

Figure 7–4 Distributed, governed UX development

This doesn't prevent refactoring, because there are still likely UX testing projects occurring on a different timeline that can alter results, but it is a fairly efficient process, and we have seen some good and consistent designs emerge in this model.

Summary

In this chapter, we described how there is far more to designing a system than simply understanding user needs. We introduced *project* and *system stakeholders* and provided guidance as to how to find them, classify them, and start to get an understanding of their expectations and specific needs. Of course, we already identified users as one such important stakeholder, but in this chapter we described how to refine that further by developing *primary* and *secondary user personas*. Designing a system to these personas will help produce a system that is inherently easier and more pleasant to use.

We've concluded with a discussion of the challenge of creating effective user experience designs in rapid, iterative development. This is indeed a nontrivial problem that agile teams have evolved various means of addressing. We've provided a couple of suggestions as to how to approach this tricky problem as well.

In the next chapters, we'll move on to other important aspects of agile team requirements practices, starting with the next chapter, Estimating and Velocity.

This page intentionally left blank

Chapter 8

Agile Estimating and Velocity

Though this be madness, yet there is method in't.

—*Hamlet*, Act 2, scene 2, Shakespeare

Introduction

One of the misperceptions about agile development is that it is a Dilbert-esque practice of software teams coding away without requirements, little or no planning, no intra-team coordination, and no documentation. Indeed, the myths of agile often prevent agile to be applied in circumstances where the benefits would otherwise be substantial. As we hope to illustrate throughout this book, nothing could be further from the truth because agile is the most disciplined and quality-driven set of development practices the industry has invented to date. But the myths remain.

When it comes to *estimating* and *scheduling* work, for example, the myth is propagated by misunderstandings of the apparent silliness (or the perceived unprofessional and unscientific nature) of the agile estimating process. Agile teams compound this because they (often rightly) refuse to make any long-term commitments about deliverables. Worse, they talk about their team's abilities in funny words such as *velocity*, *velocipacity*, *story points*, *modified Fibonacci series estimating*, and our all-time personal favorite *velocity in gummy bears per sprint*.

In the realm of agile mythology, agile *estimating* and its companion, *team velocity,* stand out as two of the oddest brethren. These also baffle outsiders, creating an unnecessary barrier between the language of the team and the language of the managers, executives, and other stakeholders who are dependent on the team's output. This is *not* helpful.

There's a Method to This Madness

Underneath that communication gap, however, is a proven estimating heuristic. Indeed, solid estimating skills are critical to an agile team's productivity and reliability.

This problem arose because the traditional project estimating means (use a work breakdown structure to identify every last task, estimate each task, add the tasks up, build a Gantt chart, and predict the cost and schedule) never actually worked for software projects. So, in a matter of self-defense, the teams invented something that does work, *at least for them*. By first understanding it and then extending it, we can also make it work for the enterprise.

In this chapter, we'll explore these two topics, *estimating* and *velocity*, to come to an understanding of just how useful and practical the method is. With such an understanding, we can put the enterprise on a path to higher reliability and predictability than we have ever had before.

The Goal Is the Same: More Reliable Estimates

Everyone wants more reliable estimates. Otherwise, there is no way to predict how much value a team can deliver in a time period. Plus, if an agile team can reliably predict what it can do in a short timebox, then we can more reliably predict what a few teams working together can deliver in the next month or two. Although that may seem like a modest goal, in fact it represents two breakthroughs that should set executive offices buzzing:

#1: Know where you are now: The agile project knows exactly where it is at every point in time because it is based on a subjective evaluation of working code (*really* working, as in coded, integrated, tested, evaluated, and running). Even without estimating, that is a breakthrough of significant proportions. A while back, Israel Gat commented the following:[1]

> I've spent over 30 years in the software development industry in increasingly high levels of management, and up until this (first agile) project and this very day (a sprint review), I have never ever really known exactly where my teams actually are on any significant project.

But the goodness doesn't end there, because once a team has mastered agile estimating, we can leverage another breakthrough:

#2: More accurately predict where you will be next: With effective agile estimating, teams have a fairly reliable predictor of what will be working in the next few weeks. Per the previous, they know what is working now, and they can predict what they will deliver next based on the *velocity* they have achieved in prior iterations.

1. Formerly of IBM Tivoli and BMC Software, now a Cutter Consortium Agile Enterprise consultant

Moreover, if we could fast-forward mentally through Part III of this book (or six months or so of agile deployment), an executive might be able to understand (exactly) where a larger program is, what software (exactly) is working today, and what software (exactly) is not, even in a program that affects dozens of teams. And, perhaps, a manager can even reliably predict where the program will be in a few weeks—perhaps even in 90 days and maybe more. But that's a subject for later chapters.

WHY ESTIMATE? THE BUSINESS VALUE OF ESTIMATING

For now, we'll get back to the first principles of agile estimating. As compared to coding and testing at least, estimating is overhead, so first we must understand why we bother to do it at all. From a lean perspective, some might even look at estimating as a form of waste. Indeed, in some kanban implementations, teams don't bother much at all with estimating. However, estimating provides substantial value added for several reasons.

- *Determining cost:* Effort is our proxy for cost. If we can't estimate effort, we can't estimate cost. If we have no way to estimate cost, then we have no reasonable basis for going about our business at all.
- *Establishing prioritization:* In Chapter 13, we'll introduce the cost of delay (CoD) as the primary prioritization factor for implementing features. Development effort predicts time, which is in turn a primary factor in cost of delay.
- *Scheduling and commitment:* It is unreasonable to ask a team to commit to delivering an unknown thing in a certain time frame, especially longer term. However, gaining commitment to near-term deliverables is important because it materially affects the planning and business objectives of our customers. It also impacts our internal customers—those who document, train, deploy, and support the system.

In other words, *estimating is the key to unlocking the ability to commit*. The ability to commit to near-term deliverables is the key to building a reliable, agile enterprise. So, there are good business reasons why we need to take a harder look at this funny agile estimating thing.

To do so, we'll take a more experiential approach in this chapter, because that is the best way to understand what the *method in't actually is*.