Robert C. Martin Series

Agile Estimating and Planning

Mike Cohn

Forewords by Jim Highsmith, Agile Practice Director, Cutter Consortium, and Gabrielle Benefield, Director, Agile Product Development, Yahoo!

Praise for *Agile Estimating and Planning*

"Traditional, deterministic approaches to planning and estimating simply don't cut it on the slippery slopes of today's dynamic, change-driven projects. Mike Cohn's breakthrough book gives us not only the philosophy, but also the guidelines and a proven set of tools that we need to succeed in planning, estimating, and scheduling projects with a high uncertainty factor. At the same time, the author never loses sight of the need to deliver business value to the customer each step of the way."

—Doug DeCarlo, author of *eXtreme Project Management: Using Leadership, Principles and Tools to Deliver Value in the Face of Volatility* (Jossey-Bass, 2004)

"We know how to build predictive plans and manage them. But building plans that only estimate the future and then embrace change, challenge most of our training and skills. In *Agile Estimating and Planning*, Mike Cohn once again fills a hole in the Agile practices, this time by showing us a workable approach to Agile estimating and planning. Mike delves into the nooks and crannies of the subject and anticipates many of the questions and nuances of this topic. Students of Agile processes will recognize that this book is truly about agility, bridging many of the practices between Scrum and ExtremeProgramming."

—Ken Schwaber, Scrum evangelist, Agile Alliance cofounder, and signatory to the Agile Manifesto

"In *Agile Estimating and Planning*, Mike Cohn has, for the first time, brought together most everything that the Agile community has learned about the subject. The book is clear, well organized, and a pleasant and valuable read. It goes into all the necessary detail, and at the same time keeps the reader's burden low. We can dig in as deeply as we need to, without too much detail before we need it. The book really brings together everything we have learned about Agile estimation and planning over the past decade. It will serve its readers well."

—Ron Jeffries, www.XProgramming.com, author of *Extreme Programming Installed* (Addison-Wesley, 2001) and *Extreme Programming Adventures in C#* (Microsoft Press, 2004)

"*Agile Estimating and Planning* provides a view of planning that's balanced between theory and practice, and it is supported by enough concrete experiences to lend it credibility. I particularly like the quote 'planning is a quest for value.' It points to a new, more positive attitude toward planning that goes beyond the 'necessary evil' view that I sometimes hold."

—Kent Beck, author of *Extreme Programming Explained, Second Edition* (Addison-Wesley, 2005)

"Up-front planning is still the most critical part of software development. Agile software development requires Agile planning techniques. This book shows you how to employ Agile planning in a succinct, practical, and easy-to-follow manner."

—Adam Rogers, Ultimate Software

"We are true believers in the Agile methods described in this book, and have experienced a substantially positive impact from their implementation and continued use. I would highly recommend this book to anyone interested in making their software development more practical and effective."

—Mark M. Gutrich, President and CEO, Fast 401k, Inc.

This page intentionally left blank

Chapter 9

Prioritizing Themes

"The indispensable first step to getting what you want is this: Decide what you want."
—Ben Stein

There is rarely, if ever, enough time to do everything. So we prioritize. The responsibility for prioritizing is shared among the whole team, but the effort is led by the product owner. Unfortunately, it is generally difficult to estimate the value of small units of functionality, such as a single user story. To get around this, individual user stories or features are aggregated into *themes*. Stories and themes are then prioritized relative to one another for the purpose of creating a release plan. Themes should be selected such that each defines a discrete set of user- or customer-valued functionality. For example, in developing the SwimStats website, we would have themes such as these:

- Keep track of all personal records and let swimmers view them.
- Allow coaches to assign swimmers to events optimally and predict the team score of a meet.
- Allow coaches to enter practice activities and track practice distances swum.
- Integrate with popular handheld computers for use at the pool.
- Import and export data.
- Allow officials to track event results and score a meet.

Each of these themes has tangible value to the users of the software. And it would be possible to put a monetary value on each. With some research we could determine that support for handheld computers is likely to result in ¤150,000 of new sales. We could compare that with the expected ¤200,000 in new sales if the next version can be used for scoring a swim meet. We could then prioritize those themes. There is more to prioritizing, however, than simply considering the monetary return from each new set of features.

Factors in Prioritization

Determining the value of a theme is difficult, and product owners on agile projects are often given the vague and mostly useless advice of "prioritize on business value." This may be great advice at face value, but what is *business value?* To provide a more practical set of guidelines for prioritizing, in this chapter we will look at four factors that must be considered when prioritizing the development of new capabilities.

1. The financial value of having the features.
2. The cost of developing (and perhaps supporting) the new features.
3. The amount and significance of learning and new knowledge created by developing the features.
4. The amount of risk removed by developing the features.

Because most projects are undertaken either to save or to make money, the first two factors often dominate prioritization discussions. However, proper consideration of the influence of learning and risk on the project is critical if we are to prioritize optimally.

Value

The first factor in prioritizing work is the financial value of the theme. How much money will the organization make or save by having the new features included in the theme? This alone is often what is meant when product owners are given the advice to "prioritize on business value."

Often, an ideal way to determine the value of a theme is to estimate its financial impact over a period of time—usually the next few months, quarters, or possibly years. This can be done if the product will be sold commercially, as, for example, a new word processor or a calculator with embedded software would

be. It can also be done for applications that will be used within the orgainzation developing them. Chapter 10, "Financial Prioritization," describes various approaches to estimating the financial value of themes.

It can be difficult to estimate the financial return on a theme. Doing so usually involves estimating the number of new sales, the average value of a sale (including follow-on sales and maintenance agreements), the timing of sales increases, and so on. Because of the complexity in doing this, it is often useful to have an alternate method for estimating value. Because the value of a theme is related to the desirability of that theme to new and existing users, it is possible to use nonfinancial measures of desirability to represent value. This will be the subject of Chapter 11, "Prioritizing Desirability."

Cost

Naturally, the cost of a feature is a huge determinant in the overall priority of a feature. Many features seem wonderful until we learn their cost. An important, yet often overlooked, aspect of cost is that the cost can change over time. Adding support for internationalization today may take four weeks of effort; adding it in six months may take six weeks. So we should add it now, right? Maybe. Suppose we spend four weeks and do it now. Over the next six months, we may spend an additional three weeks changing the original implementation based on knowledge gained during that six months. In that case, we would have been better off waiting. Or what if we spend four weeks now and later discover that a simpler and faster implementation would have been adequate? The best way to reduce the cost of change is to implement a feature as late as possible—effectively when there is no more time for change.

Themes often seem worthwhile when viewed only in terms of the time they will take. As trite as it sounds, it is important to remember that time costs money. Often, the best way to do this while prioritizing is to do a rough conversion of story points or ideal days into money. Suppose you add up the salaries for everyone involved in a project over the past twelve weeks and come up with ¤150,000. This includes the product owner and the project manager, as well as all of the programmers, testers, database engineers, analysts, user interface designers, and so on. During those twelve weeks, the team completed 120 story points. We can tell that at a total cost of ¤150,000, 120 story points cost ¤1,250 each. Suppose a product owner is trying to decide whether thirty points of functionality should be included in the next release. One way for her to decide is to ask herself whether the new functionality is worth an investment of ¤37,500 ($30 \times 1{,}250 = 37{,}500$).

Chapter 10, "Financial Prioritization," will have much more to say about cost and about prioritizing based on financial reward relative to cost.

New Knowledge

On many projects, much of the the overall effort is spent in the pursuit of new knowledge. It is important that this effort be acknowledged and considered fundamental to the project. Acquiring new knowledge is important because at the start of a project, we never know everything that we'll need to know by the end of the project. The knowledge that a team develops can be classified into two areas:

- Knowledge about the product
- Knowledge about the project

Product knowledge is knowledge about *what* will be developed. It is knowledge about the features that will be included and about those that will not be included. The more product knowledge a team has, the better able they will be to make decisions about the nature and features of the product.

Project knowledge, by contrast, is knowledge about *how* the product will be created. Examples include knowledge about the technologies that will be used, about the skills of the developers, about how well the team functions together, and so on.

The flip side of acquiring knowledge is reducing uncertainty. At the start of a project there is some amount of uncertainty about what features the new product should contain. There is also uncertainty about how we'll build the product. Laufer (1996) refers to these types of uncertainty as *end uncertainty* and *means uncertainty*. End uncertainty is reduced by acquiring more knowledge about the product; means uncertainty is reduced through acquiring more knowledge about the project.

A project following a waterfall process tries to eliminate all uncertainty about what is being built before tackling the uncertainty of how it will be built. This is the origin of the common advice that analysis is about what will be built, and design is about how it will be built. Figure 9.1 shows both the waterfall and agile views of removing uncertainty.

On the waterfall side of Figure 9.1, the downward arrow shows a traditional team's attempt to eliminate all end uncertainty at the start of the project. This means that before they begin developing, there will be no remaining uncertainty about the end that is being pursued. The product is fully defined. The rightward arrow on the waterfall side of Figure 9.1 shows that means uncertainty (about

how the product will be built) is reduced over time as the project progresses. Of course, the complete up-front elimination of all end uncertainty is unachievable. Customers and users are uncertain about exactly what they need until they begin to see parts of it. They can then successively elaborate their needs.

Contrast this view with the agile approach to reducing uncertainty, which is shown on the right side of Figure 9.1. Agile teams acknowledge that it is impossible at the start of a project to eliminate all uncertainty about what the product is to be. Parts of the product need to be developed and shown to customers, feedback needs to collected, opinions refined, and plans adjusted. This takes time. While this is occuring the team will also be learning more about how they will develop the system. This leads to simultaneously reducing both end and means uncertainty, as shown in the agile view in Figure 9.1.

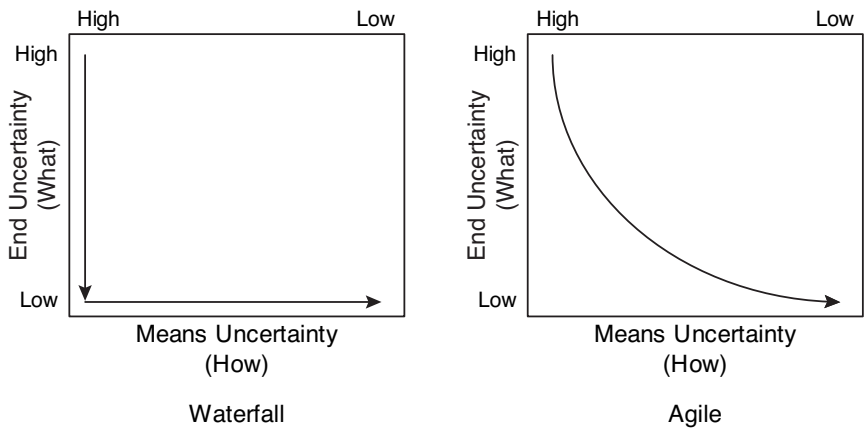

Figure 9.1 Traditional and agile views of reducing uncertainty. Adapted from Laufer (1996).

I've drawn the curve in the agile side of Figure 9.1 to show a preference toward early reduction of end uncertainty. Why didn't I draw a straight line or one that favors early reduction of means uncertainty? I drew the line as I did to reflect the importance of reducing uncertainty about what a product should be as early as possible. End uncertainty does not need to be eliminated at the outset (as hoped for in the traditional view), and it cannot be. However, one of the greatest risks to most projects is the risk of building the wrong product. This risk can be dramatically reduced by developing early those features that will best allow us to get working software in front of or in the hands of actual users.