Microsoft

BEST PRACTICES

AGILE PORTFOLIO MANAGEMENT

Jochen Krebs

Praise for *Agile Project Management*

This book is the missing link for large enterprises seeking to apply an agile approach to portfolio management.

—Mike Cohn, Author of Agile Estimating and Planning

Jochen Krebs has written a book that demystifies what happens in large organizations where various interdependencies can mystify and confuse teams making the journey to agile methods. It belongs on the bookshelves of forward-thinking executives and project managers at all levels.

—Peter Rivera, SVP, Executive Creative and Program Director, AOL Programming

This book addresses a sorely neglected area in the overall discussion of Agile methods. The solutions to many of the issues organizations face when adopting Agile methods like Scrum and XP lie in effective portfolio management, and Jochen has done well to bring this topic to the fore.

—Sanjiv Augustine, President Lithespeed, Author of Agile Project Management, Co-Founder of Agile Project Leadership Network

This is an absolute must read. Jochen simplifies a very complex concept and delivers a book that is easily read and provides a very pragmatic approach to Agile Portfolio Management.

—Robert Eagan, Director of Global Project Management Methodology for a major New York Financial Organization

Jochen Krebs' new book, Agile Portfolio Management, breaks new ground in the Agile canon by providing specific techniques for organizing work in Agile organizations at the program and portfolio level. As larger IT organizations adopt Agile broadly, many find that their legacy project selection, budgeting, and portfolio management processes are impediments to realizing the full competitive benefits their Agile development organizations can support. Joe's book will provide

organizations with some specific options for organizations to consider to increase the cadence and quality of portfolio planning and management practices to match the speed of modern Agile development shops.

—Evan Campbell, VP of Professional Services, Rally Software Development

In this unique and standard-setting book, Jochen Krebs gives the IT community a much-needed, practical, and comprehensive roadmap for creating and managing the agile portfolio.

—Doug DeCarlo, Principal, The Doug DeCarlo Group, and author of eXtreme Project Management: Using Leadership, Principles & Tools to Deliver Value in the face of Volatility

Finally, a practical Agile Project Management book for IT leadership and business stakeholders alike. Jochen's comprehensive review of Agile principles covers the financial, process, and people perspectives of three crucial vectors in portfolio management: Project, Asset, and Resource. A valuable reference book I will keep at my desk, great for Agile champions in any organization, and a must read for CIOs, PMO leaders, and product owners alike.

—Tiran Dagan, Director/Engagement Leader, Strategic Initiatives & Analysis, GE/NBC Universal

In a global, hypercompetitive marketplace, we are now forced more than ever to find ways to continuously identify, prioritize and execute software projects that routinely result in rapid deployment of compelling products and services that also routinely deliver improved enterprise innovation and profits. While Agile Software Development practices have emerged to meet this growing imperative, organizational decisioning and governance methods have remained rooted in old-school management practices that cling to big design upfront funding and execution models. Agile Portfolio Management successfully outlines a broad, practical, and well-conceived framework for aligning organizational thinking and practices that are purpose built for achieving maximum organizational agility and value creation in a change-driven world. Jochen's book is one I will ask all my CXO clients to read as they strive to better understand how best to effectively govern and exploit emerging Agile engineering practices so crucial to both survival and systematic innovation.

—Brad Murphy, CEO North America, Valtech

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2008927279

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 QWE 3 2 1 0 9 8

Distributed in Canada by H.B. Fenn and Company Ltd.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to mspinput@microsoft.com.

Acquisitions Editor: Ben Ryan
Developmental Editor: Devon Musgrave
Project Editor: Lynn Finnel
Editorial Production: Waypoint Press
Illustration by: John Hersey

Body Part No. X14-95062

To Melanie, with love and gratitude

Contents at a Glance

Table of Contents

Acknowledgments

I would like to thank the following individuals for their invaluable feedback and in-depth review during various stages of this project: Denise Cook, Marie Kalliney, and Roman Pichler. Roger LeBlanc edited this book to completion throughout many review cycles and didn't stop when I thought it was done. Steve Sagman and Lynn Finnel edited and coordinated this project and had an endless repertoire of good ideas to improve the product. The book would not be what it is without your comments and passion for this topic. Thank you all!

Completing a project like this is not possible with just pure know-how. My family and friends, especially, gave me the morale and support I needed: my Mum, Ute and Dad, Frieder; my sister,Iris ,and Rainer Löw, Markus Löw, Torben Löw, Reinhold and Sieglinde Oppenländer, Noreen and Charles Bramman, Rita O'Grady, Christopher Bramman, Gregory Bramman, Lauren, and Richard French. Another very special thank you goes to Gerhard Mauz, Markus Knierim, and Marcus Zimmermann—friends who keep me focused on what is important in life.

Introduction

If you have studied economics, you have seen the difference between tactical and strategic planning. During the 1980s, the tactical window of actions was three to five years of planning, whereas strategic planning was beyond five years, up to ten. After the information age took off during the 1990s, the rate of change with all its side effects significantly reduced this planning window, and software development cycles decreased. By now, I don't think that you will find any organization that plans in the cycles of the 1980s anymore, especially not in the information technology sector. As a matter of fact, I have heard managers tell me the joke, "Tactical is what you do today; strategy is when you plan for tomorrow." As with many jokes, some truth lies in it.

The new rate of change also affects the way we will develop systems in the future—in particular, the larger ones. The longer the project schedules, the higher the chance the scope of the project will be part of a strategy. And we know from many past experiences that a system that results from a two-year project might not do what it was intended to do. One of the reasons for this is that the planning accuracy in traditional projects is very coarse.

To address this problem, many organizations adopt or experiment with agile development—an approach based on iterative-incremental development that breaks the project's scope into smaller, manageable pieces. From managements' perspective, this approach is ideal because even the longest strategic project will have a tactical component: the iteration ahead. And it is exactly this component that will allow executives, project managers, and the project team to steer the project toward an imprecise vision. The transition to agile is not always easy, but it is the same with every change. Change bears risks and opportunities, and agile development offers many of them. This book will explore the opportunities agile presents for the development organization and its leadership, and it will address the potential risks.

One of the motivations for writing this book was to share observations I have made throughout my career, either inside or outside the project team. Many of these observations trace back to the same root cause: the lack of trust I witnessed in all aspects of an organization.

During long-term traditionally managed projects, many project status reports—especially in early phases of the project—are highly optimistic. First, it is extremely difficult for business analysts or software engineers to know that they are behind schedule. Before or during analysis, it is impossible to know how much analysis there is in total. If they don't know the total, how can they know whether they are on track? Later in the project, if the project is delayed because of new findings, the people who performed the analysis are long gone, having moved on to new projects.

Second, we were long taught that project managers are "in charge." They manage the team, give assignments, and bear responsibility for the overall success of the project. All this pressure and all these expectations are directed toward one person, who is also in charge of project communication. Given this scenario, how can we expect project communication to be neutral? Isn't it already an underlying form of mistrust for senior executives to ask for additional detailed weekly status reports? I saw these symptoms of dysfunction and mistrust over and over again in my career.

Agile portfolio management does not eliminate written communication, but it ensures that communication is based on existing agile metrics that are produced while the project is in flight. Agile portfolio management establishes a clearly defined interface between the project team and the executives, and this interface is based on agile principles. These principles are based on trust and will have a positive impact in any transition toward organizational agility.

I called the book *Agile Portfolio Management* because it is exactly the portfolio of active projects in an organization that represents the future of the enterprise—tactically and strategically, whatever your definition of those terms is. The term "agile" should highlight not only that the projects inside the portfolio are agile projects but also that the portfolio is built upon agile principles and managed dynamically. This book shows that the modern agile enterprise is transparent and that it has clearly defined communication channels; therefore, this book targets project managers and executives the same way.

This book is divided into the following three parts:

Part I: Agile for Managers

This introductory part is written for managers who want to know why agile software development has become so popular within the past several years. It explains the most important agile practices commonly applied in agile software development and agile project management.

Part II: Defining, Planning, and Measuring Portfolios

The second part is the largest section of the book. It explains practices relevant to agile portfolio management. Practices included in this section are compiling metrics, establishing a project selection process, evaluating resources, and calculating the return on investment. These chapters capture the practices of modern portfolio management.

Part III: Organization and Environment

The last part of the book provides some hands-on advice about how to orchestrate the most popular agile development processes with agile portfolio management. I highlight how Scrum-managed projects fit into the portfolio management process. In addition, this section investigates the new role of the project management office (PMO) in an agile organization.

Who Is This Book For?

This book is primarily targeting project managers, portfolio managers, business analysts, members of the PMO, and executive managers who are interested in adopting agile practices and portfolio management. I hope it will give exactly this audience an easily digestible introduction to this topic, with enough material to sponsor agility across the entire enterprise.

Although this book is not intended primarily for technical audiences, it might help them to appreciate the motivations of individuals at various levels within an organization and to better understand how factors external to the project factor into their work within the enterprise.

Beyond being a benefit to the professionals in the audience, I hope that this book will also help students currently pursuing business and science degrees to understand the importance and impact of agile development in our industry. They are the project managers of tomorrow.

Find Additional Content Online

As new or updated material becomes available that complements this book, it will be posted online on the Microsoft Press Online Developer Tools Web site. The type of material you might find includes updates to book content, articles, errata, sample chapters, and more. This Web site will be available soon at *http://www.microsoft.com/learning/books/online/developer*, and it will be updated periodically.

Support for This Book

Every effort has been made to ensure the accuracy of this book and the companion content. As corrections or changes are collected, they will be added to a Microsoft Knowledge Base article.

Microsoft Press provides support for books and companion content at the following Web site:

http://www.microsoft.com/learning/support/books/

Questions and Comments

If you have comments, questions, or ideas regarding the book or the companion content, or questions that are not answered by visiting the sites above, please send them to Microsoft Press via e-mail to

mspinput@microsoft.com

Or via postal mail to

Microsoft Press
Attn*: Agile Portfolio Management* Editor
One Microsoft Way
Redmond, WA 98052-6399

Please note that Microsoft software product support is not offered through the above addresses.

Chapter 2

Agile Software Development

In this chapter, I will define what agile software development stands for and look at the key practices of agile development from a manager's perspective. I'll tie these practices to the motivations introduced in the previous chapter.

Definitions

Let's take a look at the agile lingo before we dive into the practices of agile software development.

What Is Agile?

An agile methodology is a framework for software engineering that embraces change. For example, software development is often complex, and requirements are, especially in the beginning of a project, unknown or ambiguous. Therefore, an agile framework must have built-in mechanisms to allow the project to tackle and reduce these uncertainties. These mechanisms are listed as the key practices covered in this chapter. If only one of these key practices is left out, an agile project is incomplete.

But agile also means that the framework itself is flexible and adapts to any situation. That means that the same framework applied to two projects will have two different interpretations. The agile approach is best described as *empirical* because the process itself must be adapted to its environment, which is not necessarily the organization but could be the individual project. Agile is not a "one size fits all" solution.

Agile Processes

The following agile processes have been successfully adopted in the past in a wide range of industries. Publications, training courses, and a substantial user community exist for each of the processes listed. During this introductory part of the book, written for managers, I'll provide a quick definition of each of the popular agile processes and information about their origin. Later, in Part 3 of this book, we'll take an in-depth look at Scrum as an example of a popular agile project management process.

Extreme Programming

Extreme Programming (XP) was developed by Kent Beck, Ward Cunningham, and Ron Jeffries during the 1990s as a set of dynamic programming practices. Today, XP is the most often adopted agile methodology in the high-technology industry. The most noticeable practices of XP are pair programming and test-driven development, which we will discuss in more detail later in this chapter. Although XP provides planning practices for project management, it is often seen as an agile engineering process.

Scrum

Ken Schwaber and Jeff Sutherland, who developed Scrum in the 1990s, define it as a framework for agile project management rather than an agile process. Scrum has its origins in lean manufacturing (which is described later in this section), iterative-incremental development, and the Smalltalk engineering tools. Scrum provides a simple set of rules. Aside from these simple rules, Scrum is extremely flexible and adaptive to emerging situations. *Scrum* is a term used in rugby and not an acronym. In a nutshell, Scrum is quite different from existing project management practices. First, the role of a traditional project manager is shared among three different roles: the product owner, the scrum master, and the team. Second, two different backlogs are used to manage scope: the product backlog, which captures the scope of the product, and the sprint backlog, which contains the detail work for the current iteration. A sprint, which is the Scrum synonym for an iteration, is four weeks long. The entire Scrum team meets daily for 15 minutes so that each member can give other team members a quick update. Scrum is very popular in the agile industry. For that reason, I have dedicated Chapter 11 to providing a more detailed overview of the framework—in particular, when it is applied in the context of agile portfolio management.

Dynamic Systems Development Method

Also developed in the mid-1990s, the Dynamic Systems Development Method (DSDM) has its roots in Rapid Application Development (RAD), an iterative-incremental process model that uses prototypes at each stage of development. Compared with agile development, which strives for working software at the end of each iteration, the prototypes might be incomplete and not functioning. The prototypes do provide, however, a great way of including all stakeholders early in the requirements work, because the prototypes might be enough to get feedback—for example, from end users. Prototypes can be created for all aspects of the system, including its architecture. In reality, they are often used with graphical user interfaces. DSDM consists of the following nine principles:

- Active user involvement
- Addressing business needs
- Baselining of high-level scope

- Communication and collaboration among all stakeholders
- Frequent delivery
- Team decision making
- Integrated testing
- Iterative-incremental development
- Reversible changes throughout development

Lean Development

Lean development, originated by Bob Charette, applies the principles of lean manufacturing to software development. The result is a kit of 22 *tools*. The names of these tools still reflect their manufacturing origin—for example, "eliminate waste." Mary and Tom Poppendieck are leading advocates of lean development in the agile software development industry, having spoken and written extensively about it.

Unified Process

Listing the Unified Process (UP) here as an agile development process is not entirely correct. Compared with the other processes, the Unified Process is a descriptive process rather than an empirical one. That means the UP describes in text, like a hyperlinked version of a book, what particular roles are required to do, when they do it, and how they do it. As with any book, changes to it are more challenging to redistribute. Although the Unified Process is based on architecture-centric, iterative-incremental development principles, the project phases, disciplines, and relationships between roles and responsibilities provide less flexibility. On the other hand, it is exactly the somewhat fixed description of the process that appeals to large organizations that need to comply with a variety of standards and need to outline, for example, a companywide policy for a software development process. Nonetheless, the UP provides a tremendous step forward from the traditional waterfall process (large, separated, and sequenced software engineering phases) and is often an intermediate step between traditional processes and agile processes. There are two noteworthy flavors of the Unified Process:

- **IBM Rational Unified Process (RUP)** The RUP is a result of merging three methodologies (Booch, Objectory, and OMT) during the early and mid-1990s into one unified approach. After Rational was acquired by IBM in 2003, an eclipse-based[1] process authoring tool called the IBM Rational Method Composer (RMC) was developed. The eclipse framework provides a consistent platform for all toolmakers to develop and deploy software products. By using such a framework, developers can organize different

[1] www.eclipse.org

tools under one umbrella and view them through *perspectives*. Therefore, RUP can easily be modified using the RMC.

- **OpenUP** OpenUP is an open-source process released in October 2006. A year earlier, IBM Rational donated a significant amount of its "RUP for Small Projects" to the eclipse community (which you can learn more about at http://www.eclipse.org/epf) with the goal of developing a simple iterative-incremental process framework for small projects. Like the commercial version of RUP, OpenUP comes with an eclipse-based free process authoring tool called the Eclipse Process Framework (EPF).

Crystal Clear

The father of the crystal clear process is Alistair Cockburn. Crystal clear, like the other agile processes, is rooted in iterative-incremental development. Alistair's work is also heavily focused on humans, invention, and the idea of developing software as a corporate game. Interesting in this process is the emphasis of product and methodology fine-tuning at the beginning and in the middle of the iteration. This approach enables every project to evolve not only the system deliverables but also the chosen process itself. The ceremony of project documentation is also delegated to the project level.

Crystal clear also requires the team members to be in close proximity, the identification of real users, and that the team use basic code-versioning tooling. A major differentiator from other processes is the consideration of project communication. The larger the team, the more communication has to be factored in. Whereas a team of 5 or 10 members might easily fit into a team room with very effective communication channels, this setup might not be applicable for a team of 50 or more members. Based on the size of the team, different flavors of the crystal clear family can be adopted.

Agile Manifesto

The agile manifesto (which you can read at *http://www.agilemanifesto.org*) is the result of a meeting at the Snowbird ski resort in Utah in 2001. Prior to that date, the individual agile processes were referred to as *lightweight*. I think it was a good idea renaming it to *agile* because *lightweight* could have given the impression that it is easy to do and that heavy things were left out. Lightweight could also lead someone to believe that it was incomplete. Once you implement agile development practices, you will see that doing so can actually be difficult and that agile practices are not incomplete. The word *agile*, however, presents a challenge for the community. Nobody likes to admit that they are not agile, so some developers who call themselves agile are, in reality, not agile in our definition of the practices.

Despite any small misgivings about the name, all 17 participants at the ski resort defined the process and signed the manifesto, which was to become the measure of agility in the years to come. I remember the release of the manifesto, which immediately gave the industry a

tangible definition of agile and ground rules for adding new ideas in the future. Still today, the manifesto provides clear direction and is used to discuss and compare agile methodologies, including those in this book. More important in my opinion, the manifesto provides one common roof for all agilists, whatever their favorite agile methodology might be. Here are the core values of the manifesto:

- Individuals and interaction take precedence over processes and tools.
- Working software takes precedence over comprehensive documentation.
- Customer collaboration takes precedence over contract negotiation.
- Responding to change takes precedence over following a plan.

Please note that the left side of each statement is valued more than the right side. What is important and often misunderstood is that the manifesto does not recommend neglecting the values of the right side—for example, project documentation. It simply means that the values on the left are valued more highly. Every agile project team has to find the right balance as a team, but also they must find balance within the organization. Figure 2-1 illustrates a value system for a sample project or organization when it interprets the agile manifesto. In Figure 2-1, the arrows indicate how strictly or loosely the values on the left are balanced compared with the values on the right. But even if the arrow is placed toward the right end, it does not imply that the values on the left are overruled. It means that the organization needs to consider other elements as well.

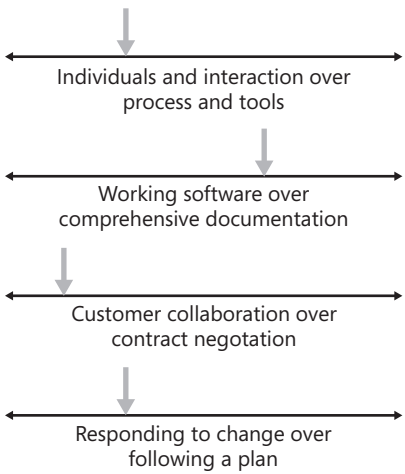

FIGURE 2-1 Example of a value system for an agile project

By the end of 2007, more than 4,700 professionals across the information technology (IT) industry had agreed to and signed the manifesto. Among the 17 authors of the manifesto were representatives from the Scrum, Extreme Programming, DSDM, and crystal clear methodologies.

Agile Alliance

The goals of the Agile Alliance (which you can read more about at *www.agilealliance.org*) are to promote agile development in the software industry. With approximately 4,000 members in December 2007, the Agile Alliance is the largest nonprofit community of agile professionals in the industry. The alliance promotes agile methodologies that comply with the agile manifesto, offers funds to members who promote agile development, offers a library of agile publications, and announces local events in the industry. The flagship of the Agile Alliance is the yearly *Agile* conference. This five-day conference offers several programs, depending on your interest, and a variety of speakers. The interest in this conference seems boundless. Attendance increased from 675 participants in 2005 to 1,100 in 2006 and to more than 1,600 in 2007. Every year, the conference is sold out, and it seems the topics create a degree of interest limited only by the size of the venue. I agree, large does not necessarily mean better quality, but I believe the attendance records demonstrate that agile development has become a mainstream approach.

Agile Project Leadership Network

Just as the Agile Alliance represents the agile manifesto, so does the Agile Project Leadership Network (APLN) represent the *project management declaration of interdependence* (PMDOI). The core values of the PMDOI are as follows:

- We increase return on investment by making continuous flow of value our focus.
- We deliver reliable results by engaging customers in frequent interactions and shared ownership.
- We expect uncertainty and manage for it through iterations, anticipation, and adaptation.
- We unleash creativity and innovation by recognizing that individuals are the ultimate source of value and creating an environment where they can make a difference.
- We boost performance through group accountability for results and shared responsibility for team effectiveness.
- We improve effectiveness and reliability through situation-specific strategies, processes, and practices.

The APLN is a nonprofit organization consisting of a community of project leaders organized into local chapters. During the remainder of this book, I'll connect the principles of the agile manifesto with the principles of the PMDOI. Furthermore, I'll apply these principles in the context of portfolio management in Part 2.

Key Practices of Agile Development

"I am breaking my two-year projects into two major phases: requirements and coding. Does that make my project agile?" The answer to this question is clearly no. Beyond that simple answer, we need to deliver an explanation of what will give a project an agile spin. The various agile processes have some successful patterns in common, which are isolated as key practices. These key practices are so essential to any agile project that they will affect the energy, spirit, and eventually the success of the agile project. These practices are, in no particular order:

- Iterative-incremental development.
- Test-driven development.
- Continuous integration.
- Face-to-face communication.

Let's explore each of the key practices in more detail in the following subsections.

Iterative-Incremental Development

This practice is, from a management perspective, the most noticeable change toward an agile approach. Instead of executing every software development cycle (requirements, design, programming, and so on) only once in the entire project (which is the traditional or waterfall way of doing it), iterative development enforces repetition and semi-parallelism of the development activities. That means the activities are extremely narrow and close to one another. They seem to blend, and the order of activities can change. That might sound strange, but you will see later in this chapter why this is a very good idea.

As a rule of thumb, the shorter the iterations, the better. That's why agile processes require a time frame for each iteration that is typically between 2 and 6 weeks.

The second aspect of this practice is the actual increment. Whereas the *iteration* provides the rhythm for the project, the *increment* demonstrates the actual progress of the project, as illustrated in Figure 2-2. Viewed in that way, it looks as if the project progresses by stepping up a staircase.

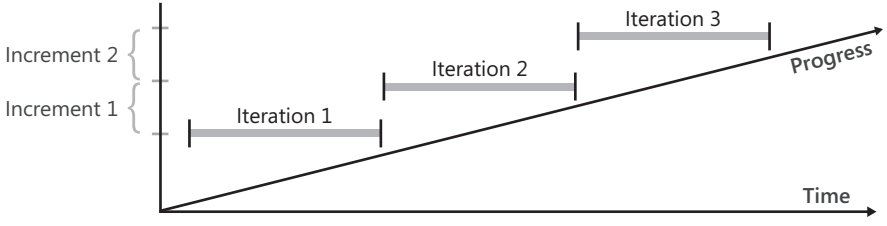

FIGURE 2-2 Incremental development and progress

In agile development, progress is measured in working software. For example, a project that focuses on a throwaway prototype during the first iteration and defining requirements in the next iteration has not demonstrated progress, based on the definition of agile. Similar to a pizza, a project must be cut in slices. The difference is, however, that we won't know in the beginning how many slices we will need and what kind of pizza it will be. We will figure it out as the project continues, slice by slice.

To state it another way, a project will have a few very high-level requirements, and the project team will take one of these requirements (for example, the highest-priority item first) and drill down into more detail. Then the team approaches the more detailed requirements, writes unit tests (as described in the "Test-Driven Development" section), designs and programs the code, and executes test cases periodically. At the end of the iteration, one high-level requirement, a slice of the project, is completed. Now, this is where iterative-incremental development really shines. One requirement is converted into working software and can be demonstrated to the customer just two weeks after the project was kicked off. Think about where your waterfall project would be two weeks into the project. Even better, by taking this approach, we opened an early feedback loop to the customer, who can now make changes and help navigate the project in the right direction (as illustrated in Figure 2-3) according to their expectations.

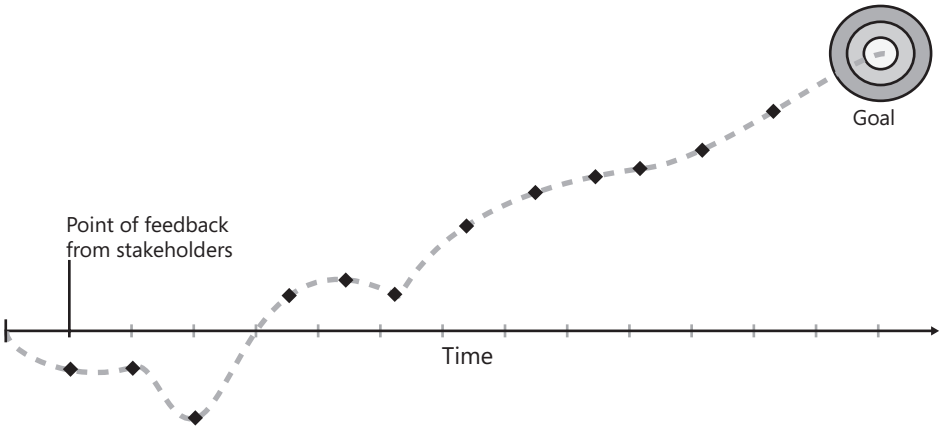

FIGURE 2-3 Feedback and direction for an iterative-incremental project

In my experience, I've seen many projects implement an iterative-incremental approach, but the majority of the customers have not taken advantage of this feedback loop. We have told so many users (wrongly) in the past that software requirements have to be complete and clearly defined up front that we will have to bring customers back to the natural way of building applications—that is, back to an agile way. The feedback loop has huge potential for many projects and shows that transitioning to agile development is a process that includes the customer as well.

Within each iteration, stakeholders will shape the direction of the increment for the scope of the iteration. In between iterations, more significant (and less disruptive) adjustments to the scope of the project are possible and often welcome. Stakeholders who are not involved in the progress of the project on a day-to-day basis are required to take part between iterations. That way, the development team also gets important feedback from peripheral stakeholders and can meet their expectations as well. According to authors Gause and Weinberg (mentioned in Chapter 1, "Motivations"), this is also called the "Yes, but..." syndrome. Whatever follows the "*but*..." are changes in the direction of refining the project. The shorter the cycles between the input and the feedback from stakeholders the better. That is one of the motivations to keep the iterations very short.

We will make heavy use of this feedback loop when agile projects are part of the portfolio.

Besides being able to better manage expectations, the customers and the project team also benefit from iterative-incremental development in many different ways. Nothing is more rewarding than hearing, iteration by iteration, that the team did a good job. Even if they did not, negative feedback allows the team to rectify things in the next iteration or even within the current one. In contrast, in the traditional approach, the verdict comes in so late that the project team cannot take corrective actions. Iterative-incremental development also overcomes requirements paralysis and the issue of ending up with too few requirements. It might sound surprising, but agile, in some respects, introduces more discipline into the process than other project methodologies—such as the discipline to deliver working software in two-week increments. What customer does not like that?

Other advantages of iterative-incremental development include the following:

- The highest risks can be reduced or eliminated in early iterations.
- Confidence in planning and estimation increases iteration by iteration.
- Based on past iterations, trends for a completion date can be determined.
- Completed means completed, not 90 percent done.
- Morale is increased through constant feedback.

Iterative-incremental development is a tremendous change to the traditional development processes currently in place in most organizations. However, this degree of change is worth undertaking, as it immediately provides invaluable benefits.

Test-Driven Development

I know this might make a lot of executives nervous, but many organizations have a lot of untested software code in production. To be fair, though, not even the software engineers are aware of that. One of the reasons is that developers write their code, debug it, make corrections to it, and so forth. What emerges is a network of classes, objects, and methods

with a variety of conditions and loops. Nested within these clauses are statements that might never be executed because the conditions that guard (and protect) these statements from execution will never be fulfilled. But who says the condition will go unfulfilled forever?

Agile development promotes a practice of test-driven development (TDD). The idea behind this practice is that the developer writes the unit test prior to the actual code. If you don't know what to test, how do you know what to code? The result is a unit test that tests the actual object but does not interfere with the object itself. The test object contains messages for all the various guards and conditions and makes sure the object acts as planned.

In agile projects, these unit tests are commonly automated, including the code coverage. That way, the project team can monitor the number of classes and the number of unit tests. If the unit tests are falling behind the number of classes, somebody on the team is not practicing test-driven development and unverified code might have entered the code base.

Test-driven development will have a significant positive impact on the quality of a project. The test cases evolve alongside the iterations; therefore, the code base that exists after each iteration is tested. Remember the waterfall approach, where the writing and execution of test cases are commonly performed at the end of the project? And that is exactly when budgets get tight and resources begin transitioning out to other projects.

The idea behind TDD is to capture the unit test code in its own component, separated from the component that implements the functionality. The activity of writing the test code and the functionality are almost parallel, with the test code slightly ahead of the game. Often the unit test as well as the functionality is developed by the same developer (pair). Then both components are compiled and the unit test executes behavior on the component. The results either pass or fail and are recorded. Through the separation of the test and the functionality, a build and, eventually, the release can be easily assembled by simply leaving the test object behind. That also means that the component does not need to be recompiled, which assures that the component with the last time-stamp is the component that passed the test.

With the test cases stored parallel to the actual code using an agile approach, it takes only a small step to automate the execution of these unit tests. This is exactly what agile developers do by using tools to execute the tests when certain triggers occur.

But even if teams are more casual about the timing of writing unit tests, tremendous organizational value can still be achieved. Once the behavior of the team has changed to make writing unit tests a common practice, the team can make a relatively easy transition to the final step—writing them prior to the actual code.

Without any unit tests, regression testing would be extremely challenging and tedious, if not impossible. The automation of the unit tests, and therefore regression testing, must be the basis for test-driven development, which will bring us to the next topic—continuous integration.

Continuous Integration

The integration of components and the testing that goes along with it is nothing new for software engineers. Where agile development substantially differs is in its continuous approach. One of the major issues with the traditional approach is that the testing of the integration is deferred to late phases in the project, and the architecture is expected to just fall into place. In reality, projects can blow up right at the worst possible moment, when little project budget and time are left. Patching problems with workarounds and mediocre approaches might help get the system out the door, but new issues are just waiting to pop up again. There has to be a better way.

One approach with iterative-incremental development is to integrate items at the end of each iteration, but that means that you have this peak and stress at the end of each iteration. So why not, instead, implement ongoing integration? That is exactly what agile teams do. Figure 2-4 shows steps in a continuous integration approach. All of these steps are commonly automated.

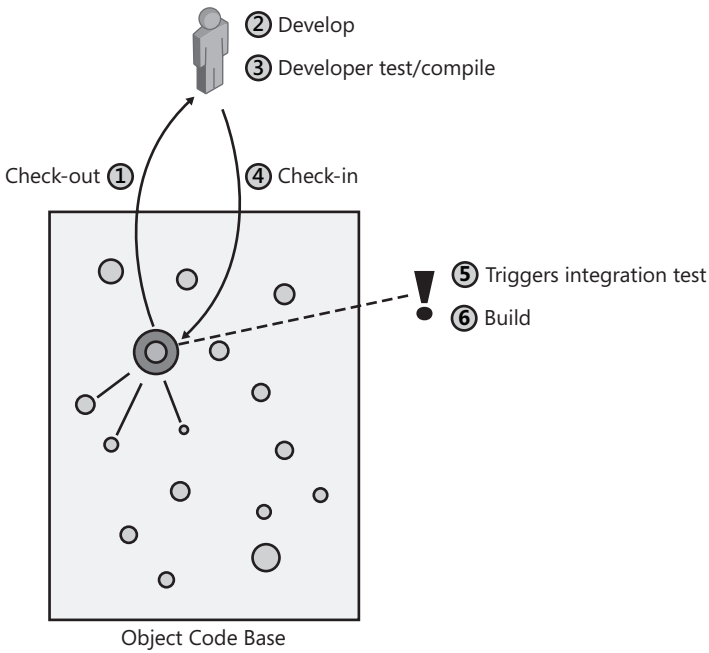

FIGURE 2-4 Continuous integration

Even the trigger for performing the integration can be linked to regular software engineering activities—for example, checking code in through a configuration management system (subversion or CVS [Concurrent Versioning System]). With a continuous integration infrastructure in place, the team, management, and executives in the organization know what the most recent successful version of the system is. When a developer on a waterfall project reports

100 percent done for a certain requirement, that means something totally different than when an agile developer reports something as being 100 percent done. In agile projects, *100 percent* means developed, tested, and integrated.

Continuous integration is relatively easy to achieve from a technical perspective. Many open-source tools have been developed to support the agile development team with its continuous integration efforts. Once the environment is in place, source and version control, execution of tests, creation of software builds, and the continuous deployment of builds are delegated to reliable tools.

Face-to-Face Communication

When I began working with IT projects, there was no such thing as electronic communication. Yes, there was e-mail, which was used to distribute information to the masses, but project team members were usually located in close proximity to one another, if not in the same room. Although we did not have the alternatives we have today, there is something about verbal and nonverbal communication that electronic communication cannot replace. Think about foreign accents or body language, for example. Agile development factors in this loss in communication and forces team members to collaborate directly with one another, person to person, without electronic firewalls. Especially when business analysts and software developers work together, face-to-face communication is essential. Anonymously sharing requirements documents just opens the door to misinterpretation and misunderstandings—not to mention that written information travels so much slower than verbal communication. Just to be clear, I'm not saying that no requirements are documented—they are co-developed, clarified, or negotiated between team members. Last but not least, we are all humans, and a project is a social network. We need these networks to increase morale and fun. I have never come across a project team that worked in an exclusively distributed way and actually had fun.

Things You Observe in an Agile Project

If we compare our agile toolkit with a menu in a restaurant, the foregoing key practices are entrees. In addition, many agile projects make heavy use of complementary tools (side dishes) that either support or enhance the impact of the key practices. Although they seem optional, they should not be ignored.

Pair Programming

Two people work together at the same time on the same code using one keyboard. Pair programming has often had a stigma attached to it in traditional organizations. "Why would I pay two people to do the job of one?" Well, in reality two people often do the job of three!

In the past, I've taught developers in training courses for Java or Smalltalk. During the course, I offered every student the opportunity to pair up with another student during the exercises. Some thought it was a good idea; others argued that they did not pay good money to not have access to the keyboard all the time. Of course, I did not force anybody either way; I just offered the option. I admit, originally the idea was a bit selfish, because I could not assist 10 engineers with individual feedback all the time. It turns out that some of the pairs helped each other with the most basic questions and needed less attention from me. Because they did work out so many things together, they used the instructor time for the tougher questions. At the end of the week, the pairs completed much more course work than the individuals. As a matter of fact, the individuals often held the pairs up in their learning goals. And as a positive side effect, the pairs usually had more fun and shared more laughs.

One of the secrets of pair programming is psychological. Adults, especially, learn most effectively by doing rather than listening or reading. The knowledge is internalized by the learner the first time someone explains or demonstrates it to that person. That is exactly what happens in pair programming. One person does a thing the other person might not know about. Through the question and answer process, the expert explains and the listener receives the knowledge. Now, equipped with the knowledge, the former listener practices it, and so forth. As a positive side effect, two people then know the same code. Think about the benefits of that in terms of sick days, vacation time, and other employee fluctuation.

Building pairs to increase productivity is not new. As a matter of fact, the army uses pairs of soldiers especially during battle at the front. Two soldiers together are more likely to stand up to an attack than individuals. Police departments also often operate with pairs of officers assigned to a case. That reduces slack but it also provides support in the event of a critical situation.

Daily Stand-Up Meetings

Short daily meetings are commonly practiced in Scrum projects, which we will focus on in a later chapter. The art of these projects is that no other meetings are scheduled besides these daily stand-up meetings, which should not last longer than 15 minutes. If you have ever tried to keep a meeting with 5 to 8 people in the 15-minute range, you know how difficult it can be. In Scrum, for example, every team member provides answers to three questions only:

- What have you done since the last meeting?
- What are you planning on doing until the next meeting?
- What issues and impediments are you facing that prevent you from accomplishing these things?

What an agile project manager is interested in are the deliverables and tangible results. That is, in my experience, the point where the daily stand-up meeting most frequently

fails. Because we are used to weekly or monthly status reports, we like to elaborate on accomplishments to justify our existence in the project. But seriously, how much tangible output can someone produce in eight hours of work? But these small accomplishments demonstrate daily progress to other team members. Also, quite an important difference from other, traditional, status meetings is the fact that the team members report to each other, not to the project manager.

Stories About Requirements

Stories on index cards are a very common practice in extreme programming. They can be estimated and planned for, and picked up by pairs of developers. Using the following template

<as a > *I want* <the feature> *so that I can* <business value>

the team can produce requirement cards and post these cards in the team room. Once reviewed and prioritized, the cards are subject to iteration planning, and team members sign up to test and develop the items on the cards. Stories just seem to have the right style and are the right size for agile teams to work with requirements.

Writing these stories is not as easy as it might first look. Many project teams struggle with coming up with the right size for the story. Some stories are too granular and user interface focused (for example, "As a customer, I want an OK button so that I can save the changes made to my account information"); other stories are too broad (for example, "As a customer, I want to order a product online so that I can save time going to the store myself").

Team Rooms

The majority of agile projects are based in a single team room, which makes face-to-face communication very easy. In the past, project teams posted their index cards on the wall or on whiteboards and signed up for pair programming by simply looking around the room. Even today, with more tools and technology available to facilitate agile projects, the team room still plays an integral part. Instead of physical index cards posted on the wall, stories are captured on electronic "cards" and projected against the wall or against whiteboards turned into smart-boards with more features to store results more quickly and more conveniently. The idea behind all this innovation, however, is the same. The entire project team always shares and works with the same information and collaboratively works toward the same goal. Therefore, electronic communication across continents is less efficient than communication among team members in the same room. In addition to overcoming difficulties in conducting the daily meetings, geographically distributed projects are challenged by meeting the goals of continuous integration, stakeholder feedback, iteration planning, and pair programming.

Frequent Releases

A *build* is a collection of tested and integrated software components that can be released either internally for verification or externally in a production environment. A build process—which includes the compilation, testing, and integration of all software—can either pass or fail. When a build passes, it represents the last good build. That build is something a test engineer, business analyst, or other stakeholder can work from and use as a test against the original story. A *release*, on the other hand, is a good build that has enough functionality to be released—internally or externally. For example, consider a project that is estimated to last for 12 months. During the first phase, a group of stories is aligned to create a release after 8 months of the project. The second phase of the project (months 9–12) include the other logical group of requirements. The first release can be used inside the organization or unleashed externally. The concept of builds and releases has tremendous influence on agile portfolio management, which I'll discuss in the second part of this book.

Self-Organized Teams

When you observe an agile project, you'll notice, for example, that a pair of developers agree during the daily meeting to tackle a story card—taking ownership and responsibility for a story card for at least a few hours or until the story is implemented. They refer to themselves as *self-managed* or *self-organized*. Surprisingly, at the end of each stand-up meeting, you'll notice that almost magically the work got divided and distributed without any top-down commands. During the next stand-up meeting, team members will report their progress to one another, which means they are also controlling one another. The traditional command-control paradigm we are so used to working in has shifted to a team-organized approach. Does that mean a manager is not needed anymore in an agile project? Because the team is already self-managed, all it needs is a *leader*. The person in that role will drop duties and responsibilities that the team has already taken over, but he or she will pick up new challenges, especially those related to project leadership. Here are just a few examples:

- Facilitating the iteration retrospectives
- Prioritizing the stories for the project and iteration with the business
- Estimating requirements
- Communicating the progress of the project
- Removing organizational and technical obstacles for the team

You might argue that these are responsibilities project managers already had in the past. Yes, but please remember, the leader's sole responsibility is to inject the vision into the project and keep the team focused without a command-and-control style. It sounds like an easy job for future project managers in agile projects, but reality shows that the opposite is the case. Good leaders can steer a project without the team feeling that it is being led. Team members

can just do their jobs while the leader creates an environment where everyone feels that everybody is working toward the same goals.

Summary

This chapter covered the most fundamental principles of agile development. These are iterative-incremental development, test-driven design, continuous integration, and face-to-face communication. Additionally, we discussed the techniques that enable agile development—for example, requirements captured in stories, the use of team rooms, and the use of pair programming. After reading this chapter, you should have a sense of the culture in an agile development team and have a vision of how these practices tackle the most common challenges in today's businesses.

Now that we've viewed the world through the glasses of an agile developer, we'll look at agility from a different angle in the following chapter—from the perspective of the agile project manager.

If you are interested in drilling down into more detail in one of these key practices, you will find cross-references for additional recommended reading in the bibliography section at the end of this book.

Chapter 3

Project Management

During the first two chapters, agile software development was introduced from a manager's perspective and we discussed why organizations are getting more and more interested in agile development. Following agile principles requires a rethinking of the traditional project management style as well. In this chapter, I'll showcase why certain project management practices need to be removed or adjusted while new agile management practices need to be acquired. These new skills are necessary for a true agile organizational transformation. Therefore, this chapter focuses on two major topics: the challenges with traditional project management in an agile environment, followed by a new agile-based definition of the role of project manager. Only the correct interpretation of agile development and agile project management will enable agile portfolio management. This chapter will be our bridge to the next part of the book.

Traditional Project Management

Project management is a well-established discipline across all industries. I once met a senior project manager who managed large enterprise information technology (IT) projects. Within his organization, he was extremely successful and had a great reputation. As it turned out, he had actually very little knowledge of information technology. His background was in manufacturing.

First I thought that, because of the size of the projects, his position was so elevated that knowledge in IT might not be necessary. But then he told me that he had experiences with all sizes of IT projects. The keys of his success were that he trusts the team and focuses on leadership and stakeholder management. He also gave me a clear understanding of why project management is a discipline on its own, which is not necessarily dependent on knowledge inside a particular industry. Too many insights can even block the real view of the project, he explained.

I thought about my meeting with him for a long time. I was skeptical and tried to imagine myself in a situation where I was, for example, managing a project in the construction industry. I then realized that even though agile project management was far from being an established concept, he had created his own recipe for project management. Years later, it did not surprise me that agile project management was founded on very similar principles: trust, shared ownership, and stakeholder collaboration. Ironically, the Project Management Institute (PMI), which takes a lot of heat from the agile community, promotes project management as a cross-industry discipline. The issue is that several concepts from the PMBOK

are difficult to endorse when applied in an agile context. The PMBOK is short for the *Project Management Body of Knowledge*, which is the framework released by the PMI. According to this process, a project steps through clearly defined and separated phases. This proves to be extremely challenging for IT projects, where requirements and expectations frequently change.

The following are four typical deliverables that are commonly created in traditional projects and promoted through the PMBOK:

- Work-breakdown structures
- Gantt charts
- Critical path analyses
- Project reports

Let's take a look at them in more detail and see what challenges they might present if applied in an agile context.

Work-Breakdown Structures

In traditional development processes, breakdown structures are hierarchical decompositions, often used in the context of work. With a work-breakdown structure (WBS), work is broken down into activities and tasks, and then these work items are broken down into more detail. (See Figure 3-1.) Sometimes cost-breakdown structures and product-work-breakdown structures are created in parallel to the WBS, always using the same approach, which is to drill down into more detail from the top down.

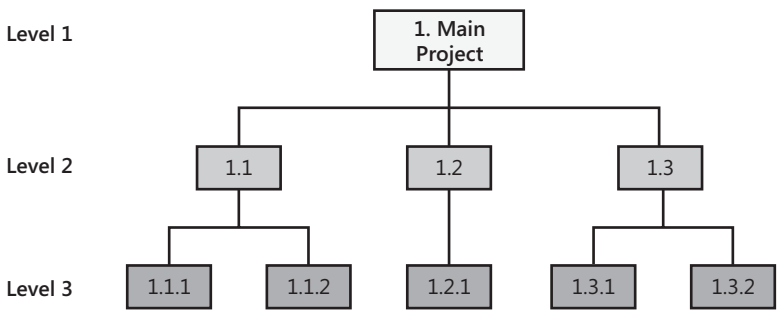

FIGURE 3-1 Work-breakdown structure (WBS)

The lowest level in the WBS hierarchy is also called a *work-package*, which is the level at which estimation is performed and assignments are done. The levels higher in the hierarchy are always combinations of the lower elements. So the bigger the project scope, the more branches there are in the WBS. The challenge with work-breakdown structures traces back to the problem of producing at an early stage a so-called master plan of work. Any work that is

not listed in the WBS won't be estimated. As a result, any work performed that is not in the WBS will produce cost overruns. We could argue that the WBS could be constantly updated to reflect the changes caused by scope changes, but that is hardly done in practice. The structure is difficult to adjust.

Another issue with estimating work up front is that the development team has not had the chance to work together and verify the estimates. If one estimate is significantly off, there is a chance that the other estimates are also out of line. Corrections of estimates while the project is under way are difficult to incorporate.

From a psychological perspective, the WBS is owned and managed by the project manager. Work is assigned to team members, and completion is controlled. This top-down, command-and-control management style does not work with agile teams, and it violates the principle of using a team-managed and team-organized approach.

I've met project managers who have told me that they will need the WBS for their own benefit, even though they won't tell the agile team about it. These managers were so accustomed to this work product that they continued using it. It is hard to be against something that gives the project manager confidence, but if the team is not using it, why bother? Are the costs of creating and maintaining a WBS for a project manager justified? Shouldn't the agile project manager work on something else and make better use of her time? Shouldn't she work on something more important and relevant for the project team—for example, removing organizational impediments or facilitating a planning session?

Instead of seeing the project from the perspective of "work to be performed," the agile team takes a different view. They look at the features and functionality of the system and document them as stories on index cards. Because the tasks in every iteration are the same, the focus of work is the story card. The activities for the iterations run parallel to one another; there is no requirements iteration or testing iteration in agile development. The activities are merged together and are continuously performed in parallel. The cards are estimated and prioritized, and the ones found relevant for the business are assigned to iterations. Remember, the feedback loop established through iterative-incremental development easily allows for this.

Gantt Charts

A Gantt chart illustrates the orchestration of the tasks according to timing and dependencies. In traditional projects, this format is frequently used to depict the project schedule. (See Figure 3-2.)

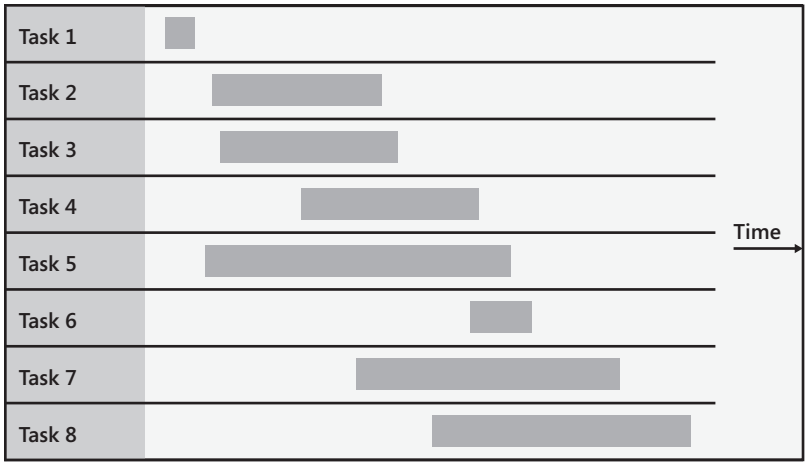

FIGURE 3-2 A Gantt chart

Very much like the WBS, the Gantt chart lays out the project schedule up front. The problem is the lack of predictability and accuracy of the schedule, especially for tasks late in the project. The planning window with the highest accuracy is the short time frame ahead of the day of planning. That is why iterations are so intuitive for many managers. Imagine a project manager in the beginning of a new long-term project. Ask him what the scope of work will be in iteration 20? It seems almost ridiculous to expect the manager to know, right? It is human nature that we think about the first iteration, the second iteration, and maybe a rough sketch of iteration 3. Iteration 3 will be refined once we start with iteration 2, and so forth.

Agile teams, however, know the project goals and have a rough estimate of how to get there. They know where the features and requirements stand at any given moment and the amount of progress expected to be achieved in an iteration.

Try the following experiment the next time you meet with a traditional project team. Ask for the project schedule, which is most likely shown as a Gantt chart. Then compare the actual day (the present day) with the plan. Ask a team member, "Based on this plan, you are currently working on task XY, correct?"

Based on my experience, the answer is most commonly an excuse for why the team is not working on that task, such as the schedule being outdated. The team's good spirit and best instincts are directed most of the time to working dynamically with the project manager instead of following a plan. Developers follow their instinct and trust their common sense. Without those, the project schedule would collapse anyway, because there are so many tasks unaccounted for in the original schedule that just seem to pop up. I have also never seen a project team start work in the morning by looking at the latest edition of the schedule. "Based on the schedule, I should be working on XY." Call it unprofessional or undisciplined; it just hasn't happened in my world.

In all fairness, maintaining the schedule is a full-time job on its own because projects are naturally very dynamic. Occasionally, especially when executives like to get an overview of the progress of the project, it is a team effort to get the Gantt chart up to date. I call this *pastcasting* (instead of *forecasting*) because the schedule's past is put back in order. For various reasons, the plan's history might be put back in order, but it provides no value to the project itself.

In agile projects, which embrace change and a dynamic way of building systems, planning with Gantt charts encounters limitations and is too expensive to maintain. Most important, such planning does not reflect what is really going on in the agile team. An agile team can tell you, however, what its plan for the next iteration is and, possibly, the plan for the following iteration also.

Critical Path Analyses

Performing a critical path analysis is a technique used to identify the shortest way through the project schedule. Any delay of any task on the critical path will delay the completion date. Figure 3-3 demonstrates that activity E cannot start before C and D are completed.

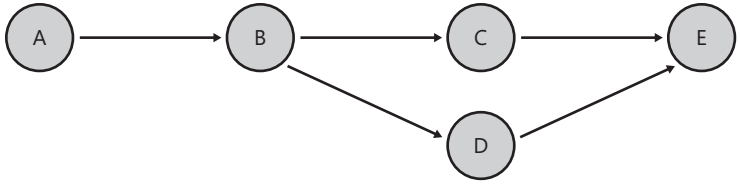

FIGURE 3-3 Task dependencies

Knowing and monitoring the critical path allows project managers to control the progress of the project. One form of illustrating the project schedule and its critical path is the activity-on-node diagram. Each node carries the following information:

- Earliest start
- Latest start
- Earliest finish
- Latest finish
- Duration
- Number
- Name of the activity

By subtracting the earliest start from the latest start, the *float* is determined. If, for example, the earliest start is day 12 in a schedule and the latest start is day 18, we have a float of 6 days. That means this task could be delayed up to 6 days without having to sacrifice the

overall schedule. Therefore, this task is not on the critical path. Every task without any float is, however, on the critical path.

Although this technique seems extremely powerful for controlling projects and puts emphasis on the critical tasks in the project, it has its shortcomings when applied in an agile project. A typical sequence for traditional project management is to have the up-front planning followed by the execution. The controlling mechanism of the critical path analysis does not include the adaptive development that takes place in an agile project. As a matter of fact, in agile projects tasks are performed in parallel—for example, writing small units of tests followed by the appropriate messages and methods in objects. A clear separation of tasks is not possible. The detailed level of the tasks would be measured in minutes rather than in hours and days, which are typically the unit of measure captured on an activity-on-node diagram.

In addition, agile projects are not considered to be late as long as the iteration is completed on time. Instead, agile teams measure how much progress has been accomplished within the last iteration. This approach creates a totally different perspective about a project's schedule.

On agile projects, the critical path is nested inside the iteration, and every iteration is on the critical path. Daily stand-up meetings expose the most critical impediments. Suppose a team planned 50 story points and completed 55. They made more progress than planned. If they again plan 50 points for the following iteration and then complete 58, the team seems to be able to deliver more than planned. Having a rough estimate for the entire project, the average progress can be used to draw conclusions on the final delivery. Having each iteration serve as feedback about progress based on tangible output seems to be more effective than projecting the completion of tasks on paper.

Project Reports

Project reports present the state of the project to stakeholders. They provide information about how the project is doing compared with its original plan. In a perfect world, a project manager can ask her team members about the progress of a task. We know by now that IT projects are often far from a perfect world. So there are basically two problems with this approach. First, the plan is often outdated and cannot be compared with reality, and second, the tasks are too broad to measure as "done" or "not done."

For example, a broad schedule might include a task of "Write User Guide," which is estimated to take 100 person-days. It is extremely difficult for a technical writer to give a status report on this task not knowing what is left to complete. A typical answer is 50 percent, 80 percent, or 90 percent done. Also, the last 10 percent might take three times as long as the first 90 percent. So how predictable is this form of status reporting?

Executives who believe in status reports might be disappointed when they realize that a project that was reporting positive figures all of a sudden has to report the breaking news, *"We are running late…"* Whoever believes that the project status reports are exclusively

created by the project manager is misinformed. Usually, status reports are a team effort that takes a lot of energy out of the entire team. But worse, while the team is providing status updates and explaining the work it has performed, team members are not making progress on the project itself.

Again, agile project teams use the increment and the iterative rhythm to report their status. For example, using two-week iterations, a project team can demonstrate its accomplishments in producing executable software. The features and requirements are either done or not done, and every claim or measurement is tangible. Using the earlier example, a team can report that a feature was not only converted to running software during an iteration, but it was also documented in the user-guide documentation. Executable software, a build, automated test cases, and a piece of documentation have been completed. All aspects of a manageable piece of functionality have been completed.

In addition, executives and stakeholders can always sneak into the daily stand-up meetings and observe the mood, progress, and issues of the project firsthand.

Summary About Challenges

By examining four typical work products found in traditional project management, you saw the challenges that arise when they are applied in the context of agile projects. The way that scope (via WBS), schedule (via Gantt charts), and progress (via progress reports) are managed and controlled (through critical-path analyses) must be adjusted to techniques that are easier to apply in the context of agile project management. The fundamental idea of project management is, however, still the same. We are still planning work, but agile projects plan around requirements, provide schedules through iterations, analyze the progress (iteratively and daily), and report on the progress to stakeholders through demonstrations and tangible progress.

Keep in mind that the challenges presented here concerning the traditional work products are based on their use in agile IT projects. In other, more predictable and reliable, industries, or in very short projects, their use and application might be appropriate. This is, however, outside my judgment.

Agile Project Management

By now, we know we want to make use of agile development practices in our IT organization, but we also know that the traditional project management techniques just do not map 1:1 to our style anymore. So let's take a look at the role of an agile project manager and that person's responsibilities and techniques for carrying out those responsibilities. Before we do that, let's look at the principles of the project management declaration of interdependence (PMDOI) to reiterate fundamental differences in agile project management.

Project Management Declaration of Interdependence

In addition to adhering to the agile manifesto, agile project managers agree to the project management declaration of interdependence. The following six core values of the PMDOI will give you a good idea of the role of an agile project manager. The PMDOI is consistent with the agile manifesto, but the core values provide a much clearer definition of agile project management. Let's see how the core values relate to the fundamental motivations for agility (which were presented in Chapter 1, "Motivations"). Please note that each core value starts with "We." Agile project managers are a community of professionals who signed the declaration. Also notice that the core values are heavily influenced by the key principle of iterative-incremental development.

- **We increase return on investment by making continuous flow of value our focus.** Instead of delivering the value of a project in one single piece, agile projects deliver in increments. Therefore, agile projects not only tackle the issues imposed by traditional project management, they also provide a cost-benefit standard as well. This is possible because the value of the project is continuously evaluated. Instead of hooraying because "We got it done!" agile project teams hooray because "The right thing is done." The "right" thing is determined by the stakeholders, who define the value of the project to the organization. The agile project manager leads the team to this goal, iteration by iteration.

  Increasing the return on investment requires feature-thinking by all participants. Features add benefits to an organization. The positive impact of some features might be stronger than others. Therefore, the sooner the benefits can be put to use, the higher the return on investment. We will see in subsequent chapters how we can schedule high-priority features to be developed early in the project to increase the return on investment.

- **We deliver reliable results by engaging customers in frequent interactions and shared ownership.** The end of an iteration is a technical and functional checkpoint for all participants of the project. Equipped with the latest successful build, the project team can demonstrate what functionality has been completed. Delivering in small intervals showcases the reliability of the project team and makes executives confident in the ability of the team. Teams and stakeholders together compare the delivered functionality with the desired outcome. After five iterations, it will be challenging for your customer to argue that the system does not meet his expectations because he agreed to significant portions of the requirements during the previous four iterations. The customer plays an integral part in these interactions and owns the direction of the project.

- **We expect uncertainty and manage for it through iterations, anticipation, and adaptation.** Remember the "Yes, but…" syndrome from the previous chapter? Your team delivered, but it did not deliver the right thing. Iterative development is the pattern used to close the feedback loop. In agile projects, the customer decides how significant

the progress of an iteration was; the customer does this through feedback, redirection, and change requests. Change is common, especially in early iterations. It is the responsibility of the project managers to steer the project toward the vision.

- **We unleash creativity and innovation by recognizing that individuals are the ultimate source of value and creating an environment where they can make a difference.** Every project team member provides value. However, sometimes team members just don't search hard enough to find their best possible role. Generally speaking, software engineers have a good work ethic and like to add value to a project. I can't recall meeting anybody at any time in my career who did not want to deliver something great. As a project manager, you should embrace interesting ideas and suggestions and create a stage for anybody to submit them. Google, for example, encourages their employees to work 20 percent of the time on projects that interest them. Great innovative solutions, such as Gmail and Google News, were the result of this policy. Make this source of ideas your strength.
- **We boost performance through group accountability for results and shared responsibility for team effectiveness.** My favorite question to pose to project teams is, "Who estimates work?" When the answer is not "We do," some work needs to be done to improve project management. Agile project managers need to be facilitators and moderators who lead the team to success. They should not be the estimators. The team estimates the work, which is prioritized by the business. It is also a team member's (or a pair's) responsibility to sign up for work. Think about this. Someone estimates work and then voluntarily signs up for it, and this initiative is combined with good work ethics. These are major ingredients for a successful delivery. Additionally, every team member reports to the rest of the team, not to the project manager.
- **We improve effectiveness and reliability through situationally specific strategies, processes, and practices.** There is a huge difference between being efficient and being effective. We can streamline a development process and make it highly efficient, but it might not be effective. Effectiveness contains results that have an impact on goals—for example, the implementation of a feature can have a cost-effective result. Knowing that effectiveness is the higher standard, the project goals can change over time, and so can the development process itself. Adapting to new situations is essential to your team's success, but the agile project manager must lead in this regard. Watching the effectiveness and reliability of the process and project are an integral part of the project manager role. As a simple example, a project team thought it would be a good idea to start the day with the daily stand-up meeting. They scheduled it for 9:00 a.m. Although penalties were assessed for late arrivals, team members had a tough time being on time. The organization had this early-arrival culture tattooed onto it, so the team had adopted it even though it was not effective for them. In the first retrospective, the team decided to have the daily stand-up meeting later in the day. They adjusted, became more effective in their meetings, and moved on.

Roles and Responsibilities

Now that you know the challenges of traditional project management and the key principles of the PMDOI, it is time to focus directly on agile project managers—who they are and what their duties are. The following roles and responsibilities define what agile project managers are.

Roles

There are primarily two roles in agile project management: the project manager and lead business analyst. Scrum has developed the de facto standard for many agile project managers. I decided to provide a definition for both terminologies.

Project Manager

The term *project management* is deeply rooted in our professional world. Even though in the agile world the project manager would be better described as a project leader, I'll go along with this widely used term. The distinction between *manager* and *leader*, however, is still important for defining the role of an agile project manager. Instead of managing projects, the project leader steers the project team and constantly shapes the vision. The definition of the Agile Project Leadership Network (APLN) reflects this fact. An agile project manager creates a team-managed platform, which encourages the values of the PMDOI. In addition, the project manager applies the key practices for agile development.

Becoming a project manager is often falsely seen as receiving a promotion. Yes, someone with special skills, which we will discuss in more detail soon, was appointed to play this role, but the role is as specialized as any other role taken on by team members. In traditional projects, the team often sees the project manager as an outsider who controls the team's progress and gives out work assignments. Good project managers who follow the agile project management approach are part of the team and are within the team's social boundary. Instead of having a command-control role, the agile project manager collaborates with the team as part of a unit.

The difference between *manager* and *leader* might appear to be subtle, but it has a significant impact on team morale. As a result, the agile project manager's role is probably best described as facilitator or moderator.

In Scrum, the scrum master assures that the rules of scrum are correctly interpreted. In this case, the scrum master executes the daily scrum meeting, conducts the retrospectives, and plans the iteration with other roles by using the project backlog. Chapter 11 takes a closer look at Scrum and how scrum principles are implemented for portfolio management. It also includes an explanation of the terms used in Scrum.

Scrum Master

In Scrum, the role of a project manager is gone. Instead of managing and leading the development effort of the project, the scrum master implements the rules of Scrum in a project and makes sure the team's path is clear so that the team can stay productive. This emphasis on *leading* becomes clear in what is known in Scrum as *sprint backlog planning*, where the project team is empowered to plan and schedule its own work. The sprint backlog contains a list of tasks to be tackled throughout the upcoming sprint, which represents in Scrum a fixed-length iteration of 30 days. The backlog contains a to-do list of activities and deliverables the team needs to tackle. Once the team has entered the sprint, the scrum master facilitates the daily scrum meeting and removes impediments for the team, such as environmental, administrative, or team-related hurdles. Especially in large organizations in which a large project spans organizational hierarchies, the scrum master must be an influential mediator and determined to facilitate the project across departmental boundaries. In such projects, the scrum master must also stand up for the team so that its requests are met; a compromised resolution is often the least desirable outcome.

The role of a scrum master translates probably best to that of a change agent or servant leader who works for the team so that it becomes more productive and effective over time. In Scrum, the responsibilities of traditional project management are divided among three different roles: the product owner, the scrum master, and of course the team.

Business Analyst

Remember the principle of shared ownership and engaging customers at frequent intervals? In internal and large-scale projects, business analysts often serve as the voice of the customer. Agile projects are so dependent on them that they are an integral part of any agile process. In an ideal world, the business analyst works full-time on the project, also inside the team boundary.

Based on my experience, defining this role in an organization is the most challenging of all. Because of the structure of the organization, the business analyst is often part of a different reporting chain. In situations like that, the business analyst is part of or reports to product management, marketing, accounting, and so on, whereas the project team reports to the chief information officer (CIO) or chief technology officer (CTO). This separation creates a gap between these two parties that has to be bridged during the project. The challenge for the project manager is to make the business analysts feel comfortable around all the technical specialists. Sharing the same project room and having direct communication channels is a major element in the agile success story.

Still today, I see so-called agile projects where requirements are handed over from the business analysts to the software development team for implementation. Organizations interested in offshore development even have to deal with language, time, and cultural barriers. In situations like this, the interpretation of agile is often just a little too loose.

Think about the difference in time and money consumed when a developer has a quick question to clarify a requirement. Asking the person on the other side of the table versus writing an e-mail message and waiting until the other person has time to reply. Often e-mail threads introduce more confusion than initially existed. Globally distributed development, where project team members work more anonymously and in a more isolated way, need wise project organization to overcome these challenges. We'll take a closer look at this topic in Part 3 of this book.

One technique, which is out of the ordinary but extremely powerful, is that the development team is allowed to work only when the business analysts are present in the room. Part-time and casual assignments immediately bounce back to the business as an impediment.

Product Owner

Scrum has a different view with regard to the business representation as well. Instead of requirements being written and clarified by team members, they are written by the product owner. This person is a highly visible figure in a scrum project and in the organization. The product owner is primarily responsible for the project's success. The product owner is also responsible for planning each release and works on the specifications for and refinement of the features for the team to develop.

The product owner sits in the driver's seat in terms of authority and prioritization of features. This role is the primary go-to person from a project portfolio management perspective, but it is also the point of contact for the entire project team when questions about the features of the current sprint emerge.

Project Team

Especially for Scrum projects, the team itself takes over many project management activities we know from traditional projects. A self-organized and empowered project team plans the work and assigns work to themselves. If the team prefers to work in pairs, team members sign up and form pairs on their own.

From a reporting perspective, the project team members derive the metrics of the project from their own daily activities, and they broadcast the metrics among the team, product owner, scrum master, and other external project stakeholders. That mechanism keeps the metrics and reports unfiltered and unmassaged.

Responsibilities

The following tasks are typically performed by an agile project manager during or in between iterations, or they are co-performed by an agile project manager and other project

team members, such as business analysts, developers, or sponsors. Most commonly the team and business analyst work hand-in-hand with the agile project manager.

Removing Impediments

Every project usually has plenty of impediments—small, unforeseen issues here and there, major problems, and organizational obstacles. Even worse, they can pop up anytime, most commonly at the worst moment. I've seen plenty of teams that have had difficulties getting started because of unknown passwords, insufficient server rights, lack of software licenses, lack of access to rooms, and other internal administrative procedures. Official escalation procedures are commonly used to resolve these issues, but even those procedures can take weeks and sometimes months to clear the issue.

In addition to administrative issues, challenges related to technologies and requirements commonly arise. There are two mechanisms in agile projects that deal with these impediments. First, the daily stand-up meeting, where issues are expressed and captured for resolution, and second, the role of the project manager (scrum master), who is responsible for tackling issues that impede team progress. Therefore, the agile project manager must be willing to go the extra mile within the organization and serve the team. You can see here that the role of a project manager can be quite different from a development role. An introverted brilliant test engineer might not like becoming an agile project manager.

During the daily meetings, the team repeatedly expresses an issue until it is resolved. In return, the project manager reports progress on resolving impediments back to the team. So everyone should have at least a little progress to report on—the team reports development activities, and the project manager reports about impediments.

If other stakeholders participate occasionally in one of the daily stand-up meetings, everyone should sense the urgency and importance of certain open issues. Just to be clear, paving the path for a project and removing impediments could be a full-time job in itself.

Iteration Planning

Iterations are usually between 2 and 6 weeks long. The shorter the better, especially in the early stages of the project. Many times, I've been asked if the iteration length can vary. In my experience, fixed-length iterations throughout the entire project work best. I have seen projects in which project managers started with two 4-week iterations and then switched to a 2-week rhythm. There are several good reasons why this can be a good idea. One reason is that it can reduce the pressure of a 2-week iteration for new agile teams. Once the team gets more familiar with iterative-incremental development, a reduction to shorter iterations might be beneficial. Changing the length of iterations (for example, 2, 3, 4, 2, and then 5 weeks) based on the intensity of the planned stories is, however, not recommended and not needed. There are great books available that assist you with assigning requirements (stories, use cases, and nonfunctional requirements) to iterations instead of the other way around. Stories

documented on index cards are common among agile developers, as are the techniques for iteration planning.

If an agile project uses use cases instead of stories, the procedure of iteration planning remains similar. A use case represents a collection of start-to-end business scenarios. Some of them are common and typical; other scenarios might be alternatives or exceptional cases and less frequently executed. The challenge with use cases is that some of them are long and complex, whereas others are easier. In addition, dependencies exist between them. If you look closer at use cases, you'll notice that they usually consist of an abundance of scenarios. Therefore, instead of tackling the entire use case, the team must focus on individual scenarios. With that approach, even dependencies between the scenarios can be resolved. A positive aspect of documenting requirements with use cases is that identifying and documenting these scenarios in this format flows naturally with the work that business analysts perform regularly. Figure 3-4 (which abbreviates *use case* as *UC*) shows how the implementation of a use case is done incrementally, scenario by scenario.

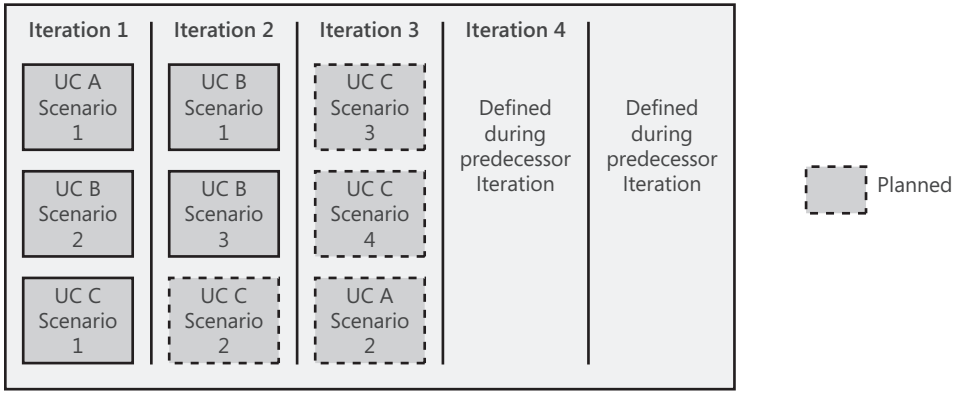

FIGURE 3-4 Iteration plan with use-case (UC) scenarios

Although a rough outline and the high-priority requirements are determined by the customer, the project manager facilitates the iteration planning session with the rest of the team. That way, dependencies between requirements can be considered and the best possible plan for the upcoming iteration can be assembled, according to the wish list of the customer.

In the second part of this book, when I outline the agile portfolio management approach, you'll see how essential iteration planning is. We will then apply the practice of iterative development to the organizational level and get the entire organization into a rhythm.

Retrospectives

Retrospectives are not lessons learned. They are much more. Traditional *lessons-learned sessions* are scheduled at the end of a project. Not only is it too late at that point for the

team to rectify behavior and remember all the "lessons," it is also challenging to get all the people at a table together when the project is over. Agile teams perform *retrospectives* in between iterations. The project manager facilitates the retrospective, which includes iteration review and iteration planning (preview). Psychologically, the term *lessons learned* also implies that there was something to learn. That is not always the case and can lead to a negative impression that something else could have been done instead by changing the project and the process.

Estimation

There are two famous questions that customers ask project managers: "How long will the project take?" and "How much will the project cost?" Especially at the beginning of a project, these questions are extremely difficult, if not impossible, to answer. If you ask a team to estimate the answers to these questions based on a few index cards or use-case scenarios, the likelihood of getting a good estimate turned around is higher. If you ask a team after a few iterations to estimate the time and cost of completing tasks on a few new cards, the accuracy of the estimate gets better and better. Why?

With every retrospective, the team reflects on the past iteration. Some requirements might have been underestimated; others might have been overestimated. The team's estimate of upcoming requirements takes the experience from previous estimations into consideration.

Traditional project managers, who have usually been transitioned from an engineering role to a project manager role, tend to produce estimates themselves or influence the estimation of others. This heroic attitude of "I could get it done in two days instead of four" does not really help if team members cannot get it done in two days. There is also no proof that the manager could get it done anyway. This kind of assistance does not add any value to the team. Sometimes project managers influence the team through a backdoor approach such as, "Do you agree that this task takes about two days?" Do estimation practices like that really help your project, your customer, and the expectations?

To prevent this issue, agile project managers stay out of the estimation exercise unless they play both roles on the team. In agile projects, the development team, and only the development team, produces the estimates. The agile project manager captures and tracks the estimation results and, of course, the actual results. Remember, the project manager should "lead" the project and connect with the customer and sponsors, rather than micromanaging people. The selected estimation technique should be easy and quick to adopt—for example, a Wide-Band-Delphi (discussed in Chapter 5, "Metrics and Reporting"). Another effective method is having team members hold up a certain number of fingers for estimated efforts and size and then averaging the results.

For example, a project team gets together and publicly estimates the effort. Everybody's input has been heard, a quick average is taken, and the process is continued for the next estimation. Even if a story was underestimated or overestimated, it will balance out over time,

just like the calls from umpires in baseball. An alternative, but more time-consuming, method is to collect the estimate from the team members anonymously. That reduces the chances that, especially during early estimating sessions, team members are unduly influenced by their colleagues.

By the way, the planning poker game (which you can learn more about at *http://www.planningpoker.com*) is a variation of the Wide-Band-Delphi estimation technique.

Reporting

As with estimating, the quality of the reporting gets better over the course of the iterations. An agile project manager has extremely powerful tools to report status almost in real time. But to keep the team focused on its deliverables, the exchange of these metrics is suggested in between iterations only. Considering that the iterations are only 2–6 weeks long, it is still a very frequent exchange of reliable information. Reporting in agile projects is extremely powerful because teams can always share internally and externally the requirements that have been completed (stories), the progress (size of stories) of the project, and the quality (defects) of the system. When asked, the project manager can accurately answer the following questions: "What has been done?" "How are we doing ?" and "How good is the system?" This is enough information to judge whether the past iteration was a success.

Remember, all the data is naturally created by the agile developers and business analysts while they are working. No extra effort is necessary to generate the data.

Daily Stand-Up Meeting

The agile project manager facilitates the daily stand-up meeting, ensures that the meeting starts on time, and ensures that it is conducted in a timely manner, usually no longer than 15 minutes. If the team is geographically dispersed, the meeting time needs to be synchronized and resources need to be synchronized and prepared. Facilitation of this meeting includes ensuring that team members report status to each other in a collegial way and side discussions are eliminated. Impediments that might be expressed during the meeting are captured as a to-do list for the project manager. The challenges of this activity are to keep the team on track and the meeting on time. To simplify the logistics of this meeting, it is usually conducted at the same time each day and within the team room. Because the daily stand-up meeting is open to the public, bystanders can receive additional ad hoc status updates from the project team.

Leading

During the iteration, leadership skills are needed to keep the team moving. Typical issues within the team are that trivial or very challenging defects have not been tackled, a build broke and no developer took ownership to resolve the issue, two developers pair up

too much and need to mix more with the rest of the team, and requirements need to be negotiated. The agile project manager reminds team members to get things done, asks for status updates for critical items throughout the day, and captures completion.

The agile project manager also represents the team to the outside world and also protects the team from the outside world. Too much interference can distract the team from its goals. Agile project managers report to the executive manager and portfolio managers, as we will see in the second part of this book.

Summary

This chapter introduced the roles and responsibilities of an agile project manager, and it explained how they differ from traditional project management. We explored typical tasks of an agile project manager—in particular, the tasks of estimating, reporting, planning, performing retrospectives, and performing day-to-day activities such as removing impediments.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Jochen Krebs

Jochen Krebs is an accomplished agile mentor and instructor. He is also the founder of Incrementor (*www.incrementor.com*), an agile coaching and training services company in New York. His focus is agile project management and requirements management, where he works directly with project management offices and their portfolios. During his 15+ year professional career, he has worked in various industries in several different roles—for example, as an object-oriented developer, project manager, instructor, consultant, and mentor. Jochen received his MSc from the Open University in Computing for Commerce and Industry.

Jochen is also the co-author of the RUP Reference and Certification Guide and has written numerous articles about agile practices in a variety of magazines. He frequently speaks at conferences and companies and spearheads the local chapter of the Agile Project Leadership Network (APLN) in New York. German-born, Jochen Krebs currently lives in Katonah, New York with his wife, Melanie.