

LEARNING MADE EASY



2nd Edition

# Agile Project Management

for  
**dummies®**

A Wiley Brand



Deliver value in weeks instead of years

Reduce risk and increase visibility of projects

Create an environment that works for agile

Mark C. Layton,  
MBA<sup>2</sup> CST, PMP, SAFe SPC  
Steven J. Ostermiller



# Agile Project Management

2nd Edition

by **Mark C. Layton and**  
**Steven J Ostermiller**

for  
**dummies®**  
A Wiley Brand

## **Agile Project Management For Dummies®, 2nd Edition**

Published by: **John Wiley & Sons, Inc.**, 111 River Street, Hoboken, NJ  
07030-5774, [www.wiley.com](http://www.wiley.com)

Copyright © 2017 by John Wiley & Sons, Inc., Hoboken, New Jersey  
Published simultaneously in Canada No part of this publication may be  
reproduced, stored in a retrieval system or transmitted in any form or by any  
means, electronic, mechanical, photocopying, recording, scanning or  
otherwise, except as permitted under Sections 107 or 108 of the 1976 United  
States Copyright Act, without the prior written permission of the Publisher.  
Requests to the Publisher for permission should be addressed to the  
Permissions Department, John Wiley & Sons, Inc., 111 River Street,  
Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at  
<http://www.wiley.com/go/permissions>.

**Trademarks:** Wiley, For Dummies, the Dummies Man logo, Dummies.com, Making Everything Easier, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and may not be used without written permission. SAFe and Scaled Agile Framework are registered trademarks of Scaled Agile, Inc. Certified Scrum Developer, Certified Scrum Product Owner, Certified Scrum Professional, Certified Scrum Trainer, and Certified ScrumMaster are registered trademarks of Scrum Alliance. PMI Agile Certified Practitioner and PMI-ACP are registered trademarks of Project Management Institute, Inc. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED

IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services, please contact our Customer Care Department within the U.S. at 877-762-2974, outside the U.S. at 317-572-3993, or fax 317-572-4002. For technical support, please visit <https://hub.wiley.com/community/support/dummies>.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit [www.wiley.com](http://www.wiley.com).

Library of Congress Control Number: 2017948508

ISBN 978-1-119-40569-6 (pbk); ISBN 978-1-119-40574-0 (ebk); ISBN 978-1-119-40573-3 (ebk)

# Agile Project Management For Dummies®

To view this book's Cheat Sheet, simply go to [www.dummies.com](http://www.dummies.com) and search for “Agile Project Management For Dummies Cheat Sheet” in the Search box.

## Table of Contents

### [Cover](#)

### [Introduction](#)

[About This Book](#)

[Foolish Assumptions](#)

[Icons Used in This Book](#)

[Beyond the Book](#)

[Where to Go from Here](#)

### [Part 1: Understanding Agile](#)

#### [Chapter 1: Modernizing Project Management](#)

[Project Management Needed a Makeover](#)

[Introducing Agile Project Management](#)

#### [Chapter 2: Applying the Agile Manifesto and Principles](#)

[Understanding the Agile Manifesto](#)

[Outlining the Four Values of the Agile Manifesto](#)

[Defining the 12 Agile Principles](#)

[Adding the Platinum Principles](#)

[Changes as a Result of Agile Values](#)

[The Agile Litmus Test](#)

#### [Chapter 3: Why Being Agile Works Better](#)

[Evaluating Agile Benefits](#)

[How Agile Approaches Beat Historical Approaches](#)

[Why People Like Being Agile](#)

## **[Part 2: Being Agile](#)**

### **[Chapter 4: Agile Approaches](#)**

[Diving under the Umbrella of Agile Approaches](#)

[Reviewing the Big Three: Lean, Scrum, and Extreme Programming](#)

[Putting It All Together](#)

### **[Chapter 5: Agile Environments in Action](#)**

[Creating the Physical Environment](#)

[Low-Tech Communicating](#)

[High-Tech Communicating](#)

[Choosing Tools](#)

### **[Chapter 6: Agile Behaviors in Action](#)**

[Establishing Agile Roles](#)

[Establishing New Values](#)

[Changing Team Philosophy](#)

## **[Part 3: Agile Planning and Execution](#)**

### **[Chapter 7: Defining the Product Vision and Product Roadmap](#)**

[Agile Planning](#)

[Defining the Product Vision](#)

[Creating a Product Roadmap](#)

[Completing the Product Backlog](#)

### **[Chapter 8: Planning Releases and Sprints](#)**

[Refining Requirements and Estimates](#)

[Release Planning](#)

[Sprint Planning](#)

### **[Chapter 9: Working throughout the Day](#)**

[Planning Your Day: The Daily Scrum](#)

[Tracking Progress](#)

[Agile Roles in the Sprint](#)

[Creating Shippable Functionality](#)

[The End of the Day](#)

## **[Chapter 10: Showcasing Work, Inspecting, and Adapting](#)**

[The Sprint Review](#)

[The Sprint Retrospective](#)

## **[Chapter 11: Preparing for Release](#)**

[Preparing the Product for Deployment: The Release Sprint](#)

[Preparing for Operational Support](#)

[Preparing the Organization for Product Deployment](#)

[Preparing the Marketplace for Product Deployment](#)

## **[Part 4: Agile Management](#)**

### **[Chapter 12: Managing Scope and Procurement](#)**

[What's Different about Agile Scope Management?](#)

[Managing Agile Scope](#)

[What's Different about Agile Procurement?](#)

[Managing Agile Procurement](#)

### **[Chapter 13: Managing Time and Cost](#)**

[What's Different about Agile Time Management?](#)

[Managing Agile Schedules](#)

[What's Different about Agile Cost Management?](#)

[Managing Agile Budgets](#)

### **[Chapter 14: Managing Team Dynamics and Communication](#)**

[What's Different about Agile Team Dynamics?](#)

[Managing Agile Team Dynamics](#)

[What's Different about Agile Communication?](#)

[Managing Agile Communication](#)

### **[Chapter 15: Managing Quality and Risk](#)**

[What's Different about Agile Quality?](#)

[Managing Agile Quality](#)

[What's Different about Agile Risk Management?](#)

[Managing Agile Risk](#)

## Part 5: Ensuring Agile Success

### Chapter 16: Building a Foundation

Organizational and Individual Commitment

Choosing the Right Pilot Team Members

Creating an Environment That Enables Agility

Support Agility Initially and Over Time

### Chapter 17: Scaling across Agile Teams

Multi-Team Agile Projects

Making Work Digestible through Vertical Slicing

Aligning through Roles with Scrum at Scale

Multi-Team Coordination with LeSS

Reducing Dependencies with Nexus

Joint Program Planning with SAFe

Modular Structures with Enterprise Scrum

### Chapter 18: Being a Change Agent

Becoming Agile Requires Change

Why Change Doesn't Happen on Its Own

Strategic Approaches to Implementing and Managing Change

Platinum Edge's Change Roadmap

Avoiding Pitfalls

Signs Your Changes Are Slipping

## Part 6: The Part of Tens

### Chapter 19: Ten Key Benefits of Agile Project Management

Better Product Quality

Higher Customer Satisfaction

Reduced Risk

Increased Collaboration and Ownership

More Relevant Metrics

Improved Performance Visibility

Increased Project Control

Improved Project Predictability

Customized Team Structures

[Higher Team Morale](#)

## **[Chapter 20: Ten Key Factors for Project Success](#)**

[Dedicated Team Members](#)

[Collocation](#)

[Automated Testing](#)

[Enforced Definition of Done](#)

[Clear Product Vision and Roadmap](#)

[Product Owner Empowerment](#)

[Developer Versatility](#)

[Scrum Master Clout](#)

[Management Support for Learning](#)

[Transition Support](#)

## **[Chapter 21: Ten Metrics for Agile Organizations](#)**

[Return on Investment](#)

[Satisfaction Surveys](#)

[Defects in Production](#)

[Sprint Goal Success Rates](#)

[Time to Market](#)

[Lead and Cycle Times](#)

[Cost of Change](#)

[Team Member Turnover](#)

[Skill Versatility](#)

[Manager-to-Creator Ratio](#)

## **[Chapter 22: Ten Valuable Resources for Agile Professionals](#)**

[Agile Project Management For Dummies Online Cheat Sheet](#)

[Scrum For Dummies](#)

[The Scrum Alliance](#)

[The Agile Alliance](#)

[The Project Management Institute Agile Community](#)

[International Consortium for Agile \(ICAgile\)](#)

[InfoQ](#)

[Lean Enterprise Institute](#)

[Extreme Programming](#)

[Platinum Edge](#)

[About the Authors](#)

[Connect with Dummies](#)

[End User License Agreement](#)

# Introduction

---

Welcome to *Agile Project Management For Dummies*. Agile project management has grown to be as common as any management technique in business today. Over the past decade and a half, we have trained and coached companies big and small, all over the world, about how to successfully run agile projects. Through this work, we found that there was a need to write a digestible guide that the average person could understand and use.

In this book, we will clear up some of the myths about what agile project management is and what it is not. The information in this book will give you the confidence to know you can be successful using agile techniques.

## About This Book

*Agile Project Management For Dummies*, 2nd Edition is more than just an introduction to agile practices and methodologies; you also discover the steps to execute agile techniques on a project. The material here goes beyond theory and is meant to be a field manual, accessible to the everyday person, giving you the tools and information you need to be successful with agile processes in the trenches of project management.

## Foolish Assumptions

Because you’re reading this book, you might have a passing familiarity with project management. Perhaps you are a project manager, a member of a project team, or a stakeholder on a project. Or perhaps you don’t have experience with formal project management approaches but are looking for a solution now. You may even have heard the term *agile* and want to know more. Or you might already be part of a project team that’s trying to be more agile.

Regardless of your experience or level of familiarity, this book provides insights you may find interesting. If nothing else, we hope it brings clarity to any confusion or myths regarding agile project management you may have encountered.

# *Icons Used in This Book*

Throughout this book, you'll find the following icons.



**TIP** Tips are points to help you along your agile project management journey. Tips can save you time and help you quickly understand a particular topic, so when you see them, take a look!



**REMEMBER** The Remember icon is a reminder of something you may have seen in past chapters. It also may be a reminder of a commonsense principle that is easily forgotten. These icons can help jog your memory when an important term or concept appears.



**WARNING** The Warning icon indicates that you want to watch out for a certain action or behavior. Be sure to read these to steer clear of big problems!



**TECHNICAL STUFF** The Technical Stuff icon indicates information that is interesting but not essential to the text. If you see a Technical Stuff icon, you don't need to read it to understand agile project management, but the information there might just pique your interest.



**ON THE WEB** On the Web means that you can find more information on the book's website at [www.dummies.com/go/agileprojectmanagementfd2e](http://www.dummies.com/go/agileprojectmanagementfd2e).

## *Beyond the Book*

Although this book broadly covers the agile project management spectrum, we can cover only so much in a set number of pages! If you find yourself at the end of this book thinking, “This was an amazing book! Where can I learn more about how to advance my projects under an agile approach?” check out [Chapter 22](#) or head over to [www.dummies.com](http://www.dummies.com) for more resources.

We’ve provided a cheat sheet for tips on assessing your current projects in relation to agile principles and free tools for managing projects using agile techniques. To get to the cheat sheet, go to [www.dummies.com](http://www.dummies.com), and then type *Agile Project Management For Dummies Cheat Sheet* in the Search box. This is also where you’ll find any significant updates or changes that occur between editions of this book.

## **Where to Go from Here**

We wrote this book so that you could read it in just about any order. Depending on your role, you may want to pay extra attention to the appropriate sections of the book. For example:

- » If you’re just starting to learn about project management and agile approaches, start with [Chapter 1](#) and read the book straight through to the end.
- » If you are a member of a project team and want to know the basics of how to work on an agile project, check out the information in [Part 3 \(Chapters 7 through 11\)](#).
- » If you are a project manager and are wondering how agile approaches affect your job, review [Part 4 \(Chapters 12 through 15\)](#).
- » If you know the basics of agile project management and are looking at bringing agile practices to your company or scaling agile practices across your organization, [Part 5 \(Chapters 16 through 18\)](#) provides you with helpful information.

# Part 1

## Understanding Agile

## **IN THIS PART ...**

Understand why project management needs to modernize due to the flaws and weaknesses in historical approaches to project management.

Find out why agile methods are growing as an alternative to traditional project management, and become acquainted with the foundation of agile project management: the Agile Manifesto and the 12 Agile Principles.

Discover the advantages that your products, projects, teams, customers, and organization can gain from adopting agile project management processes and techniques.

# Chapter 1

# Modernizing Project Management

---

## IN THIS CHAPTER

- » Understanding why project management needs to change
- » Finding out about agile project management

*Agile project management* is a style of project management that focuses on early delivery of business value, continuous improvement of the project's product and processes, scope flexibility, team input, and delivering well-tested products that reflect customer needs.

In this chapter, you find out why agile processes emerged as an approach to software development project management in the mid-1990s and why agile methodologies have caught the attention of project managers, customers who invest in the development of new software, and executives whose companies fund software development departments. This chapter also explains the advantages of agile methodologies over long-standing approaches to project management.

## ***Project Management Needed a Makeover***

A *project* is a planned program of work that requires a definitive amount of time, effort, and planning to complete. Projects have goals and objectives and often must be completed in some fixed period of time and within a certain budget.

Because you are reading this book, it's likely that you are either a project manager or someone who initiates projects, works on projects, or is affected by projects in some way.

Agile approaches are a response to the need to modernize project

management. To understand how agile approaches are revolutionizing projects, it helps to know a little about the history and purpose of project management and the issues that projects face today.

## ***The origins of modern project management***

Projects have been around since ancient times. From the Great Wall of China to the Mayan pyramids at Tikal, from the invention of the printing press to the invention of the Internet, people have accomplished endeavors big and small in projects.

As a formal discipline, project management as we know it has only been around since the middle of the twentieth century. Around the time of World War II, researchers around the world were making major advances in building and programming computers, mostly for the United States military. To complete those projects, they started creating formal project management processes. The first processes were based on step-by-step manufacturing models the United States military used during World War II.

People in the computing field adopted these step-based manufacturing processes because early computer-related projects relied heavily on hardware, with computers that filled up entire rooms. Software, by contrast, was a smaller part of computer projects. In the 1940s and 1950s, computers might have thousands of physical vacuum tubes but fewer than 30 lines of programming code. The 1940s manufacturing process used on these initial computers is the foundation of the project management methodology known as waterfall.

In 1970, a computer scientist named Winston Royce wrote “Managing the Development of Large Software Systems,” an article for the IEEE that described the phases in the waterfall methodology. The term *waterfall* was coined later, but the phases, even if they are sometimes titled differently, are essentially the same as originally defined by Royce:

1. Requirements
2. Design
3. Development
4. Integration
5. Testing

## 6. Deployment

On waterfall projects, you move to the next phase only when the prior one is complete — hence, the name waterfall.



TECHNICAL STUFF

Pure waterfall project management — completing each step in full before moving to the next step — is actually a misinterpretation of Royce's suggestions. Royce identified that this approach was inherently risky and recommended developing and testing within iterations to create products — suggestions that were overlooked by many organizations that adopted the waterfall methodology.

## SOFTWARE PROJECT SUCCESS AND FAILURE

Unfortunately, the stagnation in traditional project management approaches is catching up with the software industry. In 2015, a software statistical company called the Standish Group did a study on the success and failure rates of 10,000 projects in the US. The results of the study showed that

- *29 percent of traditional projects failed outright.* The projects were cancelled before they finished and did not result in any product releases. These projects delivered no value whatsoever.
- *60 percent of traditional projects were challenged.* The projects were completed, but they had gaps between expected and actual cost, time, quality, or a combination of these elements. The average difference between the expected and actual project results — looking at time, cost, and features not delivered — was well over 100 percent.
- *11 percent of projects succeeded.* The projects were completed and delivered the expected product in the originally expected time and budget.

Of the hundreds of billions of dollars spent on application development projects in the US alone, billions of dollars were wasted on projects that never deployed a single piece of functionality.

The waterfall methodology was the most common project management approach in software development until it was surpassed by improved approaches based on agile techniques around 2008.

## ***The problem with the status quo***

Computer technology has, of course, changed a great deal since the last century. Many people have a computer on their wrist with more power, memory, and capabilities than the largest, most expensive machine that existed when people first started using waterfall methodologies.

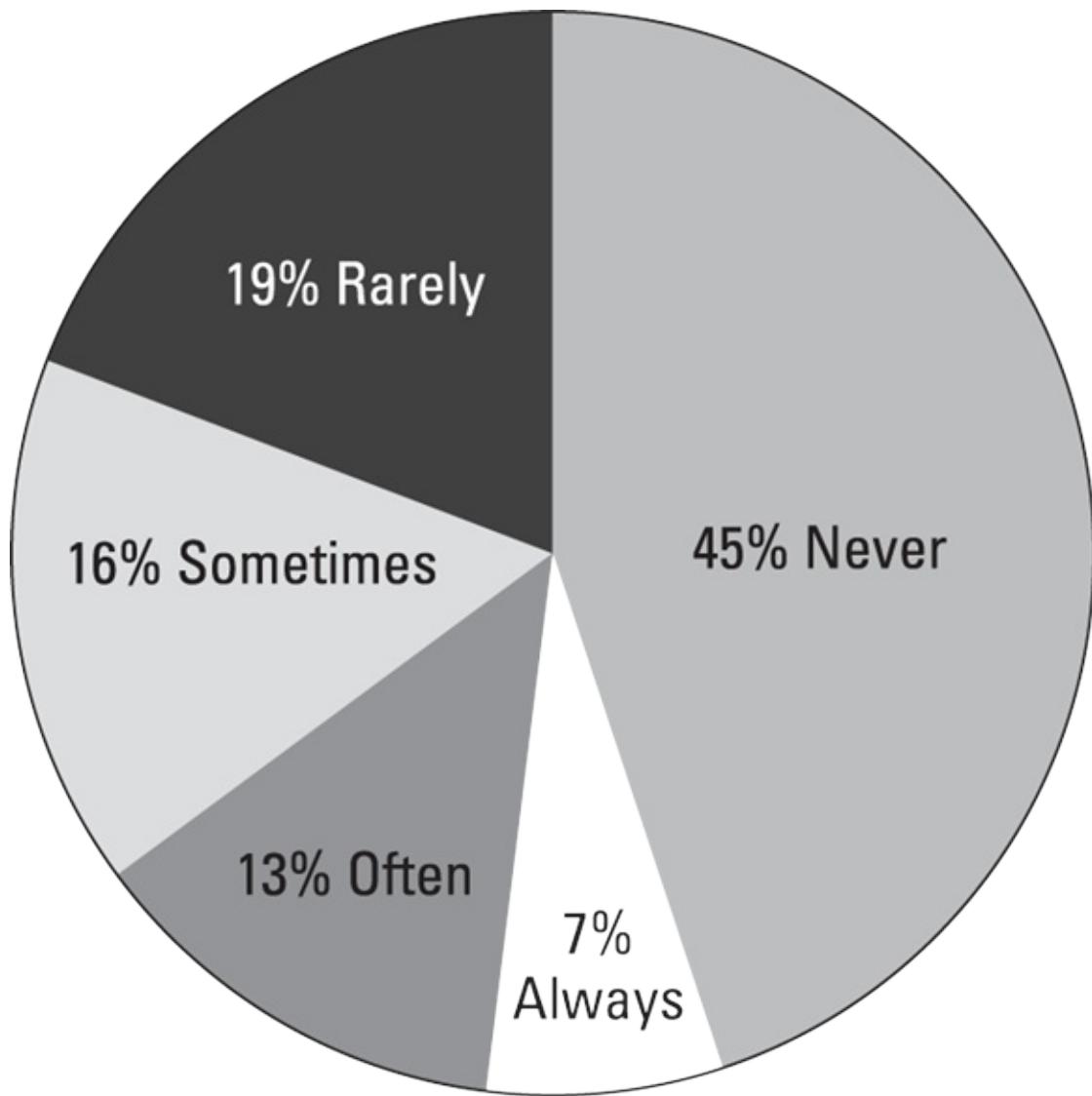
At the same time, the people using computers have changed as well. Instead of creating behemoth machines with minimal programs for a few researchers and the military, people create hardware and software for the general public. In many countries, almost everyone uses a computer, directly or indirectly, every day. Software runs our cars, our appliances, our homes; it provides our daily information and daily entertainment. Even young children use computers — a 2-year-old is almost more adept with the iPhone than her parents. The demand for newer, better software products is constant.

Somehow, during all this growth of technology, processes were not left behind. Software developers are still using project management methodologies from the 1950s, and all these approaches were derived from manufacturing processes meant for the hardware-heavy computers of the mid-twentieth century.

Today, traditional projects that do succeed often suffer from one problem: *scope bloat*, the introduction of unnecessary product features in a project.

Think about the software products you use every day. For example, the word-processing program we’re typing on right now has many features and tools. Even though we write with this program every day, we use only some of the features all the time. We use other elements less frequently. And we have never used quite a few tools — and come to think of it, we don’t know anyone else who has used them, either. The features that few people use are the result of scope bloat.

Scope bloat appears in all kinds of software, from complex enterprise applications to websites that everyone uses. [Figure 1-1](#) shows data from a Standish Group study that illustrates just how common scope bloat is. In the figure, you can see that 64 percent of requested features are rarely or never used.



Copyright 2011 Standish Group

**FIGURE 1-1:** Actual use of requested software features.

The numbers in [Figure 1-1](#) illustrate an enormous waste of time and money. That waste is a direct result of traditional project management processes that are unable to accommodate change. Project managers and stakeholders know that change is not welcome mid-project, so their best chance of getting a potentially desirable feature is at the start of a project. Therefore, they ask for

- » Everything they need
- » Everything they think they may need
- » Everything they want

» Everything they think they may want

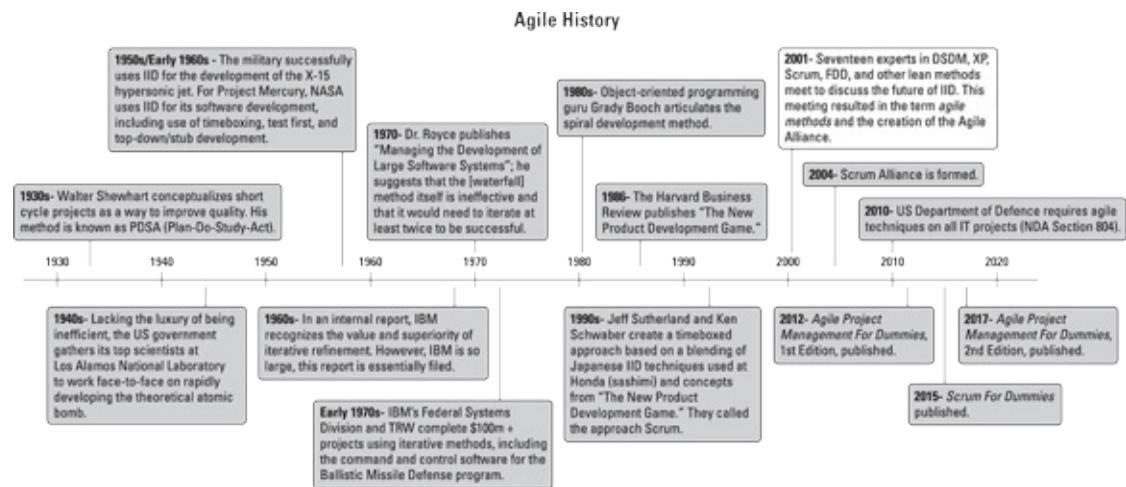
The result is the bloat in features that results in the statistics in [Figure 1-1](#).

The problems associated with using outdated management and development approaches are not trivial. These problems waste billions of dollars a year. The billions of dollars lost in project failure in 2015 (see the sidebar, “[Software project success and failure](#)”) could equate to millions of jobs around the world.

Over the past two decades, people working on projects have recognized the growing problems with traditional project management and have been working to create a better model: agile project management.

## Introducing Agile Project Management

The seeds for agile techniques have been around for a long time. In fact, agile values, principles, and practices are simply a codification of common sense. [Figure 1-2](#) shows a quick history of agile project management, dating to the 1930s with Walter Shewhart’s Plan-Do-Study-Act (PDSA) approach to project quality.



**FIGURE 1-2:** Agile project management timeline.

In 1986, Hirotaka Takeuchi and Ikujiro Nonaka published an article called “New New Product Development Game” in the *Harvard Business Review*. Takeuchi and Nonaka’s article described a rapid, flexible development strategy to meet fast-paced product demands. This article first paired the term

*scrum* with product development. (*Scrum* originally referred to a player formation in rugby.) Scrum eventually became one of the most popular agile project management frameworks.

In 2001, a group of software and project experts got together to talk about what their successful projects had in common. This group created the *Agile Manifesto*, a statement of values for successful software development:

### **Manifesto for Agile Software Development\***

We are uncovering better ways of developing

software by doing it and helping others do it.

Through this work we have come to value:

**Individuals and interactions** over processes and tools

**Working software** over comprehensive documentation

**Customer collaboration** over contract negotiation

**Responding to change** over following a plan

That is, while there is value in the items on

the right, we value the items on the left more.

\* Agile Manifesto Copyright © 2001: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas.

*This declaration may be freely copied in any form, but only in its entirety through this notice.*

These experts also created the *Principles behind the Agile Manifesto*, 12 practices that help support the values in the Agile Manifesto. We list the Agile Principles and describe the Agile Manifesto in more detail in [Chapter 2](#).

Agile, in product development terms, is a descriptor for project management approaches that focus on people, communications, the product, and flexibility. If you're looking for the agile methodology, you won't find it. However, all agile methodologies (for example, crystal), frameworks (for example, scrum), techniques (for example, user story requirements), and tools (for example, relative estimating) have one thing in common: adherence to the Agile Manifesto and the 12 Agile Principles.

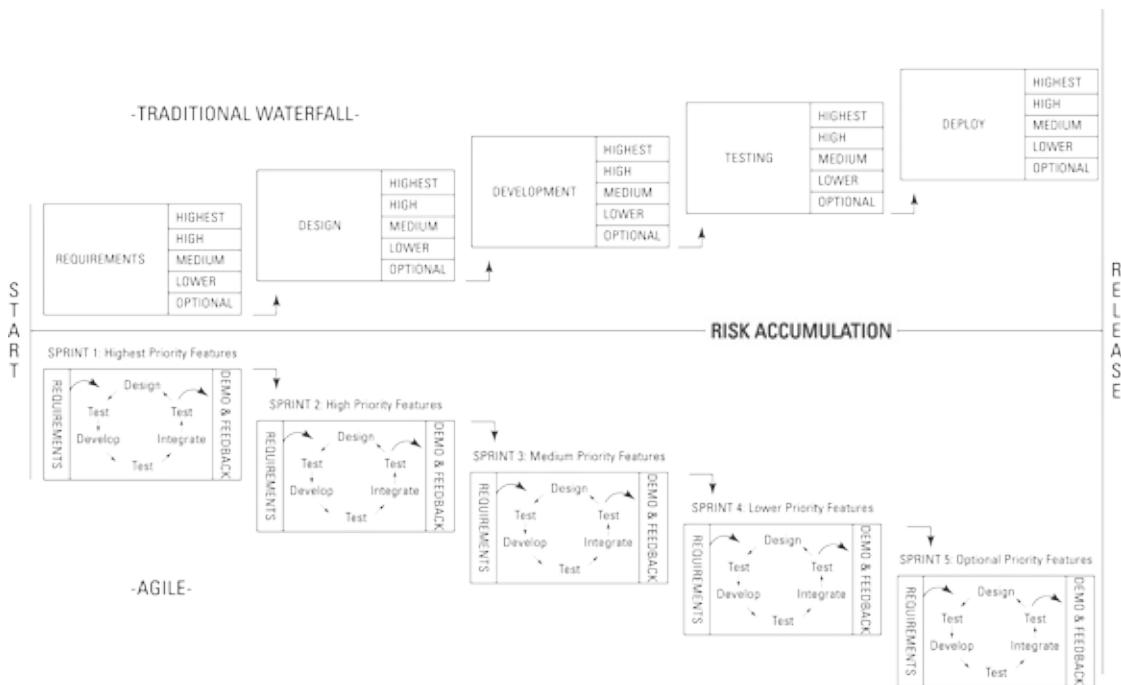
## **How agile projects work**

Agile approaches are based on an *empirical control method* — a process of making decisions based on the realities observed in the project. In the context of software development methodologies, an empirical approach can be effective in both new product development and enhancement and upgrade projects. By using frequent and firsthand inspection of the work to date, you can make immediate adjustments, if necessary. Empirical control requires

- » **Unfettered transparency:** Everyone involved in an agile project knows what is going on and how the project is progressing.
- » **Frequent inspection:** The people who are invested in the product and process the most regularly evaluate the product and process.
- » **Immediate adaptation:** Adjustments are made quickly to minimize problems; if an inspection shows that something should change, it is changed immediately.

To accommodate frequent inspection and immediate adaptation, agile projects work in *iterations* (smaller segments of the overall project). An agile project involves the same type of work as in a traditional waterfall project: You create requirements and designs, develop the product, document it, and if necessary, integrate the product with other products. You test the product, fix any problems, and deploy it for use. However, instead of completing these steps for all product features at once, as in a waterfall project, you break the project into iterations, also called *sprints*.

[Figure 1-3](#) shows the difference between a linear waterfall project and an agile project.



**FIGURE 1-3:** Waterfall versus agile project.



**WARNING** Mixing traditional project management methods with agile approaches is like saying, “I have a Porsche 911 Turbo. However, I’m using a wagon wheel on the front left side. How can I make my car as fast as the other Porsches?” The answer, of course, is you can’t. If you fully commit to an agile approach, you will have a better chance of project success.

## Why agile projects work better

Throughout this book, you see how agile projects work better than traditional projects. Agile project management approaches can produce more successful projects. The Standish Group study, mentioned in the sidebar “[Software project success and failure](#),” found that while 29 percent of traditional projects failed outright, that number dropped to only 9 percent on agile projects. The decrease in failure for agile projects is a result of agile project teams making immediate adaptations based on frequent inspections of progress and customer satisfaction.

Here are some key areas where agile approaches are superior to traditional project management methods:

- » **Project success rates:** In [Chapter 15](#), you find out how the risk of catastrophic project failure falls to almost nothing on agile projects. Agile approaches of prioritizing by business value and risk ensure early success or failure. Agile approaches to testing throughout the project help ensure that you find problems early, not after spending a large amount of time and money.
- » **Scope creep:** In [Chapters 7, 8](#), and [12](#), you see how agile approaches accommodate changes throughout a project, minimizing scope creep. On agile projects, you can add new requirements at the beginning of each sprint without disrupting development flow. By fully developing prioritized features first, you prevent scope creep from threatening critical functionality.
- » **Inspecting and adaptation:** In [Chapters 10](#) and [14](#), you find details of how regular inspections and adaptation work on agile projects. Agile project teams — armed with frequent feedback from complete development cycles and working, shippable functionality — can improve their processes and their products with each sprint.

Throughout many chapters in this book, you discover how you gain control of the outcome of agile projects. Testing early and often, adjusting priorities as needed, using better communication techniques, and regularly demonstrating and releasing product functionality allow you to fine-tune your control over a wide variety of factors on agile projects.

# Chapter 2

# Applying the Agile Manifesto and Principles

---

## IN THIS CHAPTER

- » Defining the Agile Manifesto and the 12 Agile Principles
- » Describing the Platinum Principles
- » Understanding what has changed in project management
- » Taking the agile litmus test

This chapter describes the basics of what it means to be agile: the Agile Manifesto, with its four values, and the 12 agile principles behind the Agile Manifesto. We also expand on these basics with three additional Platinum Principles, which Platinum Edge (owned by Mark) crafted after years of experience supporting organizations' agile transitions.

This foundation provides product development teams with the information needed to evaluate whether the project team is following agile principles, as well as whether their actions and behaviors are consistent with agile values. When you understand these values and principles, you'll be able to ask, "Is this agile?" and be confident in your answer.

## *Understanding the Agile Manifesto*

In the mid-1990s, the Internet was changing the world right before our eyes. The people working in the booming dot-com industry were under constant pressure to be the first to market with fast-changing technologies. Development teams worked day and night, struggling to deliver new software releases before competitors made their companies obsolete. The information technology (IT) industry was completely reinvented in a few short years.

Given the pace of change at that time, cracks inevitably appeared in

conventional project management practices. Using traditional methodologies such as waterfall, which is discussed in [Chapter 1](#), didn't allow developers to be responsive enough to the market's dynamic nature and to emerging new approaches to business. Development teams started exploring alternatives to these outdated approaches to project management. In doing so, they noticed some common themes that produced better results.

In February 2001, 17 of these new methodology pioneers met in Snowbird, Utah, to share their experiences, ideas, and practices; to discuss how best to express them; and to suggest ways to improve the world of software development. They couldn't have imagined the effect their meeting would have on the future of project management. The simplicity and clarity of the manifesto they produced and the subsequent principles they developed transformed the world of information technology and continue to revolutionize project management in every industry, not just software development.

Over the next several months, these leaders constructed the following:

- » **The Agile Manifesto:** An intentionally streamlined expression of core development values
- » **The Agile Principles:** A set of 12 guiding concepts that support agile project teams in implementing agile techniques and staying on track
- » **The Agile Alliance:** A community development organization focused on supporting individuals and organizations that are applying agile principles and practices

The group's work was destined to make the software industry more productive, more humane, and more sustainable.

The Agile Manifesto is a powerful statement, carefully crafted using fewer than 75 words:

### **Manifesto for Agile Software Development**

We are uncovering better ways of developing software by doing it and helping others do it.  
Through this work we have come to value:

**Individuals and interactions** over processes and tools

**Working software** over comprehensive documentation

**Customer collaboration** over contract negotiation

**Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

\* Agile Manifesto Copyright © 2001: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas This declaration may be freely copied in any form, but only in its entirety through this notice.

No one can deny that the Agile Manifesto is both a concise and an authoritative statement. Whereas traditional approaches emphasize a rigid plan, avoid change, document everything, and encourage hierachal-based control, the manifesto focuses on

- » People
- » Communications
- » The product
- » Flexibility

The Agile Manifesto represents a big shift in focus in how projects are conceived, conducted, and managed. If we read only the items on the left, we understand the new paradigm that the manifesto signers envisioned. They found that by focusing more attention on individuals and interactions, teams would more effectively produce working software through valuable customer collaboration and by responding well to change. In contrast, the traditional primary focus on processes and tools often produces comprehensive or excess documentation to comply with contract negotiations and to follow an unchanging plan.

Research and experience illustrate why agile values are so important:

- » **Individuals and interactions over processes and tools:** Why? Because research shows a 50 times increase in performance when we get individuals and interactions right. One of the ways we get this right is by collocating a development team with an empowered product owner.

- » **Working software over comprehensive documentation:** Why? Because failure to test for and correct defects during the sprint can take up to 24 times more effort and cost in the next sprint. And after the functionality is deployed to the market, if a production support team that wasn't involved in product development performs the testing and fixing, the cost is up to 100 times more.
- » **Customer collaboration over contract negotiation:** Why? Because a dedicated and accessible product owner can generate a fourfold increase in productivity by providing in-the-moment clarification to the development team, aligning customer priorities with the work being performed.
- » **Responding to change over following a plan:** Why? Because 64 percent of features developed under a waterfall model are rarely or never used (as discussed in [Chapter 1](#)). Starting with a plan is vital, but that is when we know the least. Agile teams don't plan less than waterfall teams — they plan as much or more. However, agile teams take a just-in-time approach, planning just enough when needed. Adaptation of the plan to the realities along the way is how agile teams deliver products that delight customers.

The creators of the Agile Manifesto originally focused on software development because they worked in the IT industry. However, agile project management techniques have spread beyond software development and even outside computer-related products. Today, people use agile approaches to create products in a variety of industries, including biotech, manufacturing, aerospace, engineering, marketing, nonprofit work, and even building construction. If you want early empirical feedback on the product or service you're providing, you can benefit from agile methods.



REMEMBER The Agile Manifesto and 12 Agile Principles directly refer to software; we leave these references intact when quoting the manifesto and principles throughout the book. If you create non-software products, try substituting your product as you read on.

## ***Outlining the Four Values of the Agile***

# **Manifesto**

The Agile Manifesto was generated from experience, not from theory. As you review the values described in the following sections, consider what they would mean if you put them into practice. How do these values support meeting time-to-market goals, dealing with change, and valuing human innovation?

## ***Value 1: Individuals and interactions over processes and tools***

When you allow each person to contribute his or her unique value to a project, the result can be powerful. When these human interactions focus on solving problems, a unified purpose can emerge. Moreover, the agreements come about through processes and tools that are much simpler than conventional ones.

A simple conversation in which you talk through a project issue can solve many problems in a relatively short time. Trying to emulate the power of a direct conversation with email, spreadsheets, and documents results in significant overhead costs and delays. Instead of adding clarity, these types of managed, controlled communications are often ambiguous and time-consuming and distract the development team from the work of creating a product.

Consider what it means if you value individuals and interactions highly. [Table 2-1](#) shows some differences between valuing individuals and valuing interactions and valuing processes and tools.

## **TABLE 2-1 Individuals and Interactions versus Processes and Tools**

	<i>Individuals and Interactions Have High Value</i>	<i>Processes and Tools Have High Value</i>
Pros	Communication is clear and effective. Communication is quick and efficient. Teamwork becomes strong as people work together. Development teams can self-organize. Development teams have more chances to innovate. Development teams can customize processes as	Processes are clear and can be easy to follow. Written records of communication exist.

necessary.

Development team members can take personal

ownership of the project.

Development team members can have deeper job satisfaction.

---

Cons	Development team members must have the <i>capacity</i> to be involved, responsible, and innovative.	People may over-rely on processes instead of finding the best ways to create good products. One process doesn't fit all teams — different people have different work styles. One process doesn't fit all projects. Communication can be ambiguous and time-consuming.
	People may need to let go of ego to work well as members of a team.	

---

---



**ON THE WEB** You can find a blank template of [Table 2-1](#) on the book's companion website at [www.dummies.com/go/agileprojectmanagementfd2e](http://www.dummies.com/go/agileprojectmanagementfd2e) — jot down the pros and cons of each approach that apply to you and your projects.



**REMEMBER** If processes and tools are seen as the way to manage product development and everything associated with it, people and the way they approach the work must conform to the processes and tools. Conformity makes it hard to accommodate new ideas, new requirements, and new thinking. Agile approaches, however, value people over process. This emphasis on individuals and teams puts the focus on their energy, innovation, and ability to solve problems. You use processes and tools in agile project management, but they're intentionally streamlined and directly support product creation. The more robust a process or tool, the more you spend on its care and feeding and the more you defer to it. With people front and center, however, the result is a leap in productivity. An agile environment is human-centric and participatory and can be readily adapted to new ideas and innovations.

## **Value 2: Working software over comprehensive documentation**

A development team's focus should be on producing working functionality.

On agile projects, the only way to measure whether you are truly finished with a product requirement is to produce the working functionality associated with that requirement. For software products, working software means the software meets what we call the *definition of done*: at the very least, developed, tested, integrated, and documented. After all, the working product is the reason for the project.

Have you ever been in a status meeting where you reported that you were, say, 75 percent done with your project? What would happen if your customer told you, “We ran out of money. Can we have our 75 percent now?” On a traditional project, you would not have any working software to give the customer — “75 percent done” traditionally means you are 75 percent in progress and 0 percent done. On an agile project, however, by using the definition of done, you would have working, potentially shippable functionality for 75 percent of your project requirements — the highest-priority 75 percent of requirements.



**REMEMBER** Although agile approaches have roots in software development, you can use them for other types of products. This second agile value can easily read, “Working functionality over comprehensive documentation.”

Tasks that distract from producing valuable functionality must be evaluated to see whether they support or undermine the job of creating a working product. [Table 2-2](#) shows a few examples of traditional project documents and their usefulness. Think about the documents produced on a recent project you were involved in.

## **TABLE 2-2 Identifying Useful Documentation**

<b>Document</b>	<b>Does the Document Support Product Development?</b>	<b>Is the Document Barely Sufficient or Gold-Plated?</b>
Project schedule created with expensive project management software, complete with Gantt Chart.	No.  Start-to-finish schedules with detailed tasks and dates tend to provide more than what is necessary for product development. Also, many of these details change before you develop future features.	Gold-plated.  Although project managers might spend a lot of time creating and updating project schedules, project team members tend to want to know only key deliverable dates. Management often wants to know only whether the project is on time, ahead of schedule, or behind.

Requirements documentation.	Yes. All projects have requirements — details about product features and needs. Development teams need to know those needs to create a product.	Possibly gold-plated; should be barely sufficient. Requirements documents can easily grow to include unnecessary details. Agile approaches provide simple ways to describe product requirements.
Product technical specifications.	Yes. Documenting how you created a product can make future changes easier.	Possibly gold-plated; should be barely sufficient. Agile documentation includes just what it needs — development teams often don't have time for extra flourishes and are keen to minimize documentation.
Weekly status report.	No. Weekly status reports are for management purposes but do not assist product creation.	Gold-plated. Knowing project status is helpful, but traditional status reports contain outdated information and are much more burdensome than necessary.
Detailed project communication plan.	No. Although a contact list can be helpful, the details in many communication plans are useless to product development teams.	Gold-plated. Communication plans often end up being documents about documentation — an egregious example of busywork.



**REMEMBER** With agile project management, the term *barely sufficient* is a positive description, meaning that a task, document, meeting, or almost anything on a project includes only what it needs to achieve the goal. Being barely sufficient is practical and efficient — it's sufficient, just enough. The opposite of barely sufficient is *gold-plating*, adding unnecessary frivolity — and effort — to a feature, task, document, meeting, or anything else.

All projects require some documentation. On agile projects, documents are useful only if they support product development and are barely sufficient to serve the design, delivery, and deployment of a working product in the most direct, unceremonious way. Agile approaches dramatically simplify the administrative paperwork relating to time, cost control, scope control, or reporting.



**ON THE WEB** You can find a blank template of [Table 2-2](#) at [www.dummies.com/go/agileprojectmanagementfd2e](http://www.dummies.com/go/agileprojectmanagementfd2e). Use that form to

assess how well your documentation directly contributed to the product and whether it was barely sufficient.



TIP We often stop producing a document and see who complains. After we know the requestor of the document, we strive to better understand why the document is necessary. The *five whys* work great in this situation — ask “why” after each successive answer to get to the root reason for the document. After you know the core reason for the document, see how you can satisfy that need with an agile artifact or streamlined process.

Agile project teams produce fewer, more streamlined documents that take less time to maintain and provide better visibility into potential issues. In the coming chapters, you find out how to create and use simple tools (such as a product backlog, a sprint backlog, and a task board) that allow project teams to understand requirements and assess status daily. With agile approaches, project teams spend more time on development and less time on documentation, resulting in a more efficient delivery of a working product.

## ***Value 3: Customer collaboration over contract negotiation***

The customer is not the enemy. Really.

Historical project management approaches usually involve customers at three key points:

- » **Start of a project:** When the customer and the project manager — or another project team representative — negotiate contract details.
- » **Any time the scope changes during the project:** When the customer and the project manager negotiate changes to the contract.
- » **End of a project:** When the project team delivers a completed product to the customer. If the product doesn’t meet the customer’s expectations, the project manager and the customer negotiate additional changes to the contract.

This historical focus on negotiation discourages potentially valuable customer

input and can even create an adversarial relationship between customers and project teams.



**WARNING** You will never know less about a product than at the project's start.

Locking product details into a contract at the beginning of your project means you have to make decisions based on incomplete knowledge. If you have flexibility for change as you learn more about a product, you'll ultimately create better products.

The agile pioneers understood that collaboration, rather than confrontation, produced better, leaner, more useful products. As a result of this understanding, agile methods make the customer part of the project on an ongoing basis.

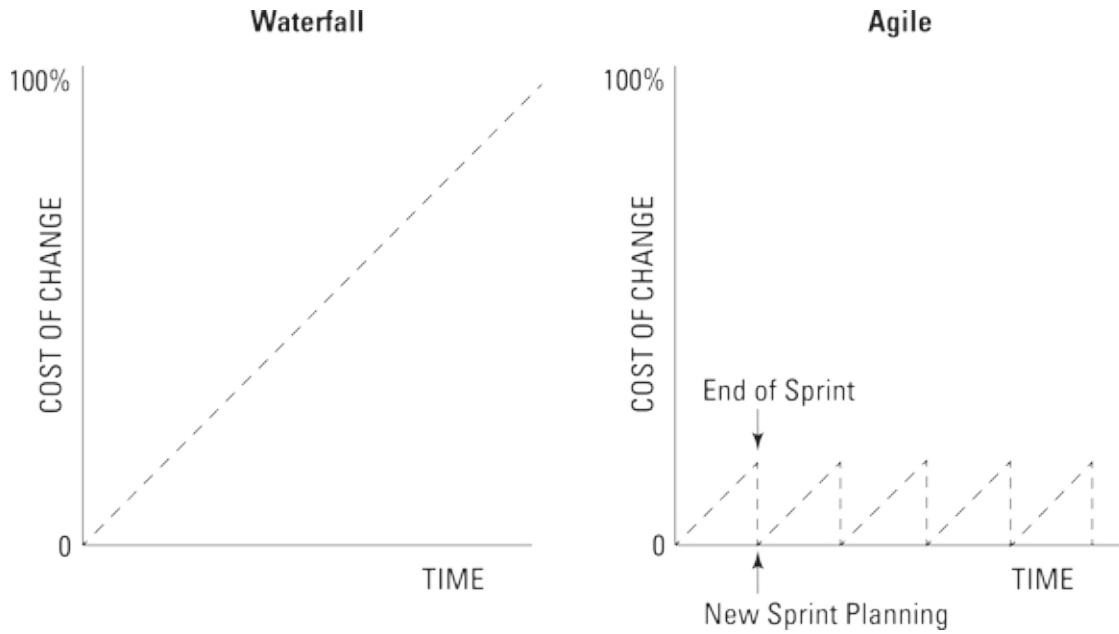
Using an agile approach in practice, you'll experience a partnership between the customer and the development team in which discovery, questioning, learning, and adjusting during the course of the project are routine, acceptable, and systematic.

## ***Value 4: Responding to change over following a plan***

Change is a valuable tool for creating great products. Project teams that can respond quickly to customers, product users, and the market are able to develop relevant, helpful products that people want to use.

Unfortunately, traditional project management approaches attempt to wrestle the change monster and pin it to the ground so it goes out for the count. Rigorous change management procedures and budget structures that can't accommodate new product requirements make changes difficult. Traditional project teams often find themselves blindly following a plan, missing opportunities to create more valuable products.

[Figure 2-1](#) shows the relationship between time, opportunity for change, and the cost of change on a traditional project. As time — and knowledge about your product — increases, the ability to make changes decreases and costs more.



**FIGURE 2-1:** Traditional project opportunity for change.

By contrast, agile projects accommodate change systematically. The flexibility of agile approaches increases project stability because change in an agile project is predictable and manageable. In later chapters, you discover how the agile approaches to planning, working, and prioritization allow project teams to respond quickly to change.

As new events unfold, the project team incorporates those realities into the ongoing work. Any new item becomes an opportunity to provide additional value instead of an obstacle to avoid, giving development teams a greater opportunity for success.

## ***Defining the 12 Agile Principles***

In the months following the publication of the Agile Manifesto, the original signatories continued to communicate. To support teams making agile transitions, they augmented the four values of the manifesto with 12 principles behind the Agile Manifesto.



**REMEMBER** These principles, along with the Platinum Principles (explained later in the “[Adding the Platinum Principles](#)” section) can be used as a litmus

test to see whether the specific practices of your project team are true to the intent of the agile movement.

Following is the text of the original 12 principles, published in 2001 by the Agile Alliance:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity — the art of maximizing the amount of work not done — is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

These agile principles provide practical guidance for development teams.

Another way of organizing the 12 principles is to consider them in the following four distinct groups:

- » Customer satisfaction
- » Quality
- » Teamwork
- » Project management

The following sections discuss the principles according to these groups.

## ***Agile principles of customer satisfaction***

Agile approaches focus on customer satisfaction, which makes sense. After all, the customer is the reason for developing the product in the first place.

While all 12 principles support the goal of satisfying customers, principles 1, 2, 3, and 4 stand out for us:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.

You may define the customer on a project in a number of ways:

- » In project management terms, the customer is the person or group paying for the project.
- » In some organizations, the customer may be a client, external to the organization.
- » In other organizations, the customer may be a project stakeholder or stakeholders in the organization.
- » The person who ends up using the product is also a customer. For clarity and to be consistent with the original 12 agile principles, in this book, we call that person *the user*.

How do you enact these principles? Simply do the following:

- » Agile project teams include a *product owner*, a person who is responsible for ensuring translation of what the customer wants into product requirements.
- » The product owner prioritizes product features in order of business value or risk and communicates priorities to the development team. The development team delivers the most valuable features on the list in short cycles of development, known as *iterations* or *sprints*.
- » The product owner has deep and ongoing involvement throughout each day to clarify priorities and requirements, make decisions, provide feedback, and quickly answer the many questions that pop up during a project.
- » Frequent delivery of working functionality allows the product owner and the customer to have a full sense of how the product is developing.
- » As the development team continues to deliver complete, working, potentially shippable functionality every four to eight weeks or less, the value of the total product grows incrementally, as do its functional capabilities.
- » The customer accumulates value for his or her investment regularly by receiving new, ready-to-use functionality throughout the project, rather than waiting until the end of what might be a long project for the first, and maybe only, delivery of releasable product features.

In [Table 2-3](#), we list some customer satisfaction issues that commonly arise on projects. Use [Table 2-3](#) and gather some examples of customer dissatisfaction that you've encountered. Do you think agile project management would make a difference? Why or why not?

**TABLE 2-3 Customer Dissatisfaction and How Agile Might Help**

<i>Examples of Customer Dissatisfaction with Projects</i>	<i>How Agile Approaches Can Increase Customer Satisfaction</i>
The product requirements were misunderstood by the development team	<p>Product owners work closely with the customer to define and refine product requirements and provide clarity to the development team.</p> <p>Agile project teams demonstrate and deliver working functionality at regular intervals. If a product doesn't work the way the customer thinks it should work, the customer is able to provide feedback at the end of the sprint, not before it's too late at the end of</p>

team.

the project.

The product wasn't delivered when the customer needed it.

Working in sprints allows agile project teams to deliver high-priority functionality early and often.

The customer can't request changes without additional cost and time.

Agile processes are built for change. Development teams can accommodate new requirements, requirement updates, and shifting priorities with each sprint, offsetting the cost of these changes by removing the lowest-priority requirements — functionality that likely will never or rarely get used.



ON THE WEB

You can find a blank template of the form at  
[www.dummies.com/go/agileprojectmanagementfd2e](http://www.dummies.com/go/agileprojectmanagementfd2e).



TIP

Agile strategies for customer satisfaction include the following:

- » Producing, in each iteration, the highest-priority features first
- » Ideally, locating the product owner and the other members of the project team in the same place to eliminate communication barriers
- » Breaking requirements into groups of features that can be delivered in four to eight weeks or less
- » Keeping written requirements sparse, forcing more robust and effective face-to-face communication
- » Getting the product owner's approval as functionality is completed
- » Revisiting the feature list regularly to ensure that the most valuable requirements continue to have the highest priority

## ***Agile principles of quality***

An agile project team commits to producing quality in every product it creates — from development through documentation to integration and test results — every day. Each project team member contributes his or her best work all the time. Although all 12 principles support the goal of quality delivery, principles 1, 3, 4, 6–9, and 12 stand out for us:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

These principles, in practice on a day-to-day basis, can be described as follows:

- » The development team members must have full ownership and be empowered to solve problems. They carry the responsibility for determining how to create the product, assigning tasks, and organizing product development. People not doing the work don't tell them how to do it.
- » With software development projects, an agile approach requires architectures that make coding and the product modular, flexible, and extensible. The design should address today's problems and make inevitable changes as simple as possible.
- » A set of designs on paper can never tell you that something will work. When the product quality is such that it can be demonstrated and ultimately shipped in short intervals, everyone knows that the product works — at the end of every sprint.
- » As the development team completes features, the team shows the product

owner the product functionality to get validation that it meets the acceptance criteria. The product owner's reviews should happen throughout the iteration, ideally the same day that development of the requirement was completed.

- » At the end of every iteration (lasting one to four weeks or less), working code is demonstrated to the customer. Progress is clear and easy to measure.
- » Testing is an integral, ongoing part of development and happens throughout the day, not at the end of the iteration.
- » On software projects, checking that new code is tested and integrates with previous versions occurs in small increments and may even occur several times a day (or thousands of times a day in some organizations, such as Google, Amazon, and Facebook). This process, called *continuous integration (CI)*, helps ensure that the entire solution continues to work when new code is added to the existing code base.
- » On software projects, examples of technical excellence include establishing coding standards, using service-oriented architecture, implementing automated testing, and building for future change.



TIP Agile approaches provide the following strategies for quality management:

- » Defining what *done* means at the beginning of the project and then using that definition as a benchmark for quality
- » Testing aggressively and daily through automated means
- » Building only the functionality that is needed when it's needed
- » Reviewing the software code and streamlining (refactoring)
- » Showcasing to stakeholders and customers only the functionality that has been accepted by the product owner
- » Having multiple feedback points throughout the day, iteration, and project

## ***Agile principles of teamwork***

Teamwork is critical to agile projects. Creating good products requires cooperation among all the members of the project team, including customers and stakeholders. Agile approaches support team-building and teamwork, and they emphasize trust in self-managing development teams. A skilled, motivated, unified, and empowered project team is a successful team.

Although all 12 principles support the goal of teamwork, principles 4–6, 8, 11, and 12 stand out for us as supporting team empowerment, efficiency, and excellence:

4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.



TIP Agile approaches focus on sustainable development; as knowledge workers, our brains are the value we bring to a project. If only for selfish reasons, organizations should want fresh, well-rested brains working for them. Maintaining a regular work pace, rather than having periods of intense overwork, helps keep team members' minds sharp and quality high.

Here are some practices you can adopt to make this vision of teamwork a reality:

- » Ensure that your development team members have the proper skills and

motivation.

- » Provide training sufficient to the task.
- » Support the self-organizing development team's decisions about what to do and how to do it; don't have managers tell the team what to do.
- » Hold project team members responsible as a single team, not individuals.
- » Use face-to-face communication to quickly and efficiently convey information.



**WARNING** Suppose that you usually communicate by email to Sharon. You take time to craft your message and then send it. The message sits in Sharon's inbox, and she eventually reads it. If Sharon has any questions, she writes an email in response and sends it. That message sits in your inbox until you eventually read it. And so forth. This type of table tennis communication is too inefficient to use in the middle of a rapid iteration.

- » Have spontaneous conversations throughout the day to build knowledge, understanding, and efficiency.
- » Collocate teammates in close proximity to increase clear and efficient communication. If collocation isn't possible, use video chat rather than email.
- » Make sure that *lessons learned* is an ongoing feedback loop. Retrospectives should be held at the end of each iteration, when reflection and adaptation can improve development team productivity going forward, creating ever higher levels of efficiency. A lessons learned meeting at the end of a project is of minimal value.  
The first retrospective is often the most valuable because, at that point, the project team has the opportunity to make changes to benefit the rest of the project moving forward.



**TIP** The following strategies promote effective teamwork:

- » Place the development team in the same location — this is called

*collocation.*

- » Put together a physical environment that's conducive for collaboration: a team room with whiteboards, colored pens, and other tactile tools for developing and conveying ideas to ensure shared understanding.
- » Create an environment where project team members are encouraged to speak their minds.
- » Meet face-to-face whenever possible. Don't send an email if a conversation can handle the issue.
- » Get clarifications throughout the day as they're needed.
- » Encourage the development team to solve problems rather than having managers solve problems for the development team.

## ***Agile principles of project management***

Agility in project management encompasses three key areas:

- » Making sure the development team can be productive and can sustainably increase productivity over long periods of time
- » Ensuring that information about the project's progress is available to stakeholders without interrupting the flow of development activities by asking the development team for updates
- » Handling requests for new features as they occur and integrating them into the product development cycle

An agile approach focuses on planning and executing the work to produce the best product that can be released. The approach is supported by communicating openly, avoiding distractions and wasteful activities, and ensuring that the progress of the project is clear to everyone.

All 12 principles support project management, but principles 2, 8, and 10 stand out for us:

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace

indefinitely.

10. Simplicity — the art of maximizing the amount of work not done — is essential.

Following are some advantages of adopting agile project management:

- » Agile project teams achieve faster time-to-market, and consequentially cost savings. They start development earlier than in traditional approaches because agile approaches minimize the exhaustive upfront planning and documentation that is conventionally part of the early stages of a waterfall project.
- » Agile development teams are self-organizing and self-managing. The managerial effort normally put into telling developers how to do their work can be applied to removing impediments and organizational distractions that slow down the development team.
- » Agile development teams determine how much work they can accomplish in an iteration and commit to achieving those goals. Ownership is fundamentally different because the development team is establishing the commitment, not complying with an externally developed commitment.
- » An agile approach asks, “What is the minimum we can do to achieve the goal?” instead of focusing on including all the features and extra refinements that could possibly be needed. An agile approach usually means streamlining: barely sufficient documentation, removal of unnecessary meetings, avoidance of inefficient communication (such as email), and less coding (just enough to make it work).



WARNING Creating complicated documents that aren't useful for product development is a waste of effort. It's okay to document a decision, but you don't need multiple pages on the history and nuances of how the decision was made. Keep the documentation barely sufficient, and you will have more time to focus on supporting the development team.

- » By encapsulating development into short sprints that last one to four weeks or less, you can adhere to the goals of the current iteration while accommodating change in subsequent iterations. The length of each sprint

remains the same throughout the project to provide a predictable rhythm of development for the team long-term.

- » Planning, elaborating on requirements, developing, testing, and demonstrating functionality occur within an iteration, lowering the risk of heading in the wrong direction for extended periods of time or developing something that the customer doesn't want.
- » Agile practices encourage a steady pace of development that is productive and healthy. For example, in the popular agile development set of practices called extreme programming (XP), the maximum workweek is 40 hours, and the preferred workweek is 35 hours. Agile projects are sustainable and more productive, especially long term.



WARNING Traditional approaches routinely feature a *death march*, in which the project team puts in extremely long hours for days and even weeks at the end of a project to meet a previously unidentified and unrealistic deadline. As the death march goes on, productivity tends to drop dramatically. More defects are introduced, and because defects need to be corrected in a way that doesn't break a different piece of functionality, correcting defects is the most expensive work that can be performed. Defects are often the result of overloading a system — specifically demanding an unsustainable pace of work. Check out our presentation on the negative effects of "Racing in Reverse" (<https://platinumedge.com/overtime>).

- » Priorities, experience on the existing project, and, eventually, the speed at which development will likely occur within each sprint are clear, making for good decisions about how much can or should be accomplished in a given amount of time.

If you've worked on a project before, you might have a basic understanding of project management activities. In [Table 2-4](#), we list a few traditional project management tasks, along with how you would meet those needs with agile approaches. Use [Table 2-4](#) to capture your thoughts about your experiences and how agile approaches looks different from traditional project management.

## TABLE 2-4 Contrasting Historical Project Management with Agile Project Management

<i>Traditional Project Management Tasks</i>	<i>Agile Approach to the Project Management Task</i>
Create a fully detailed project requirement document at the beginning of the project. Try to control requirement changes throughout the project.	Create a product backlog — a simple list of requirements by priority. Quickly update the product backlog as requirements and priorities change throughout the project.
Conduct weekly status meetings with all project stakeholders and developers. Send out detailed meeting notes and status reports after each meeting.	The development team meets quickly, for no longer than 15 minutes, at the start of each day to coordinate and synchronize that day's work and any roadblocks. They can update the centrally visible burndown chart in under a minute at the end of each day.
Create a detailed project schedule with all tasks at the beginning of the project. Try to keep the project tasks on schedule. Update the schedule on a regular basis.	Work within sprints and identify only specific tasks for the active sprint.
Assign tasks to the development team.	Support the development team by helping remove impediments and distractions. On agile projects, development teams define and pull (as opposed to push) their own tasks.



ON THE WEB

A blank template of **Table 2-4** is available at  
[www.dummies.com/go/agileprojectmanagementfd2e](http://www.dummies.com/go/agileprojectmanagementfd2e).



TIP

Project management is facilitated by the following agile approaches:

- » Supporting the development team
- » Producing barely sufficient documents
- » Streamlining status reporting so that information is pushed out by the development team in seconds rather than pulled out by a project manager over a longer period of time
- » Minimizing nondevelopment tasks
- » Setting expectations that change is normal and beneficial, not something to be feared or evaded

- » Adopting a just-in-time requirements refinement to minimize change disruption and wasted effort
- » Collaborating with the development team to create realistic schedules, targets, and goals
- » Protecting the development team from organizational disruptions that could undermine project goals by introducing work not relevant to the project objectives
- » Understanding that an appropriate balance between work and life is a component of efficient development

## ***Adding the Platinum Principles***

Through in-the-trenches experience working with teams transitioning to agile project management — and field testing in large, medium, and small organizations worldwide — we developed three additional principles of agile software development that we call the Platinum Principles. They are

- » Resist formality.
- » Think and act as a team.
- » Visualize rather than write.

You can explore each principle in more detail in the following sections.

### ***Resisting formality***

Even the most agile project teams can drift toward excessive formalization. For example, it isn't uncommon for us to find project team members waiting until a scheduled meeting to discuss simple issues that could be solved in seconds. These meetings often have an agenda and meeting minutes and require a certain level of mobilization and demobilization just to attend. In an agile approach, this level of formalization isn't required.



**WARNING** You should always question formalization and unnecessary, showy displays. For example, is there an easier way to get what you need? How does the current activity support the development of a quality product as

quickly as possible? Answering these questions helps you focus on productive work and avoid unnecessary tasks.

In an agile system, discussions and the physical work environment are open and free-flowing; documentation is kept to the lowest level of quantity and complexity such that it contributes value to the project, not hampers it; flashy displays, such as well-decorated presentations, are avoided. Professional, frank communications are best for the project team, and the entire environment has to make that openness available and comfortable.



TIP Strategies for success with resisting formality include the following:

- » Reducing organizational hierarchy wherever possible by eliminating titles in the project team
- » Avoiding aesthetic investments such as elaborate PowerPoint presentations or extensive meeting minute forms, especially when demonstrating shippable functionality at the end of a sprint
- » Identifying and educating stakeholders who may request complicated displays of work on the costs of such displays

## ***Thinking and acting as a team***

Project team members should focus on how the team as a whole can be most productive. This focus can mean letting go of individual niches and performance metrics. In an agile environment, the entire project team should be aligned in its commitment to the goal, its ownership of the scope of work, and its acknowledgment of the time available to achieve that commitment.

Following are some strategies for thinking and acting as a team:

- » Develop in pairs and switch partners often. Both pair programming (both partners are knowledgeable in the area) and shadowing (only one partner is knowledgeable in the area) raise product quality. You can learn more about pair programming in [Chapter 15](#).
- » Replace individual work titles with a uniform product developer title. Development activities include all tasks required to take requirements

through to functionality, including design, implementation (coding), testing, and documentation, not just writing code or turning a screwdriver.

- » Report at the project team level only, as opposed to creating special management reports that subdivide the team.
- » Replace individual performance metrics with project team performance metrics.

## ***Visualizing rather than writing***

An agile project team should use visualization as much as possible, whether through simple diagrams or computerized modeling tools. Images are much more powerful than words. When you use a diagram or mockup instead of a document, your customer can relate better to the concept and the content.

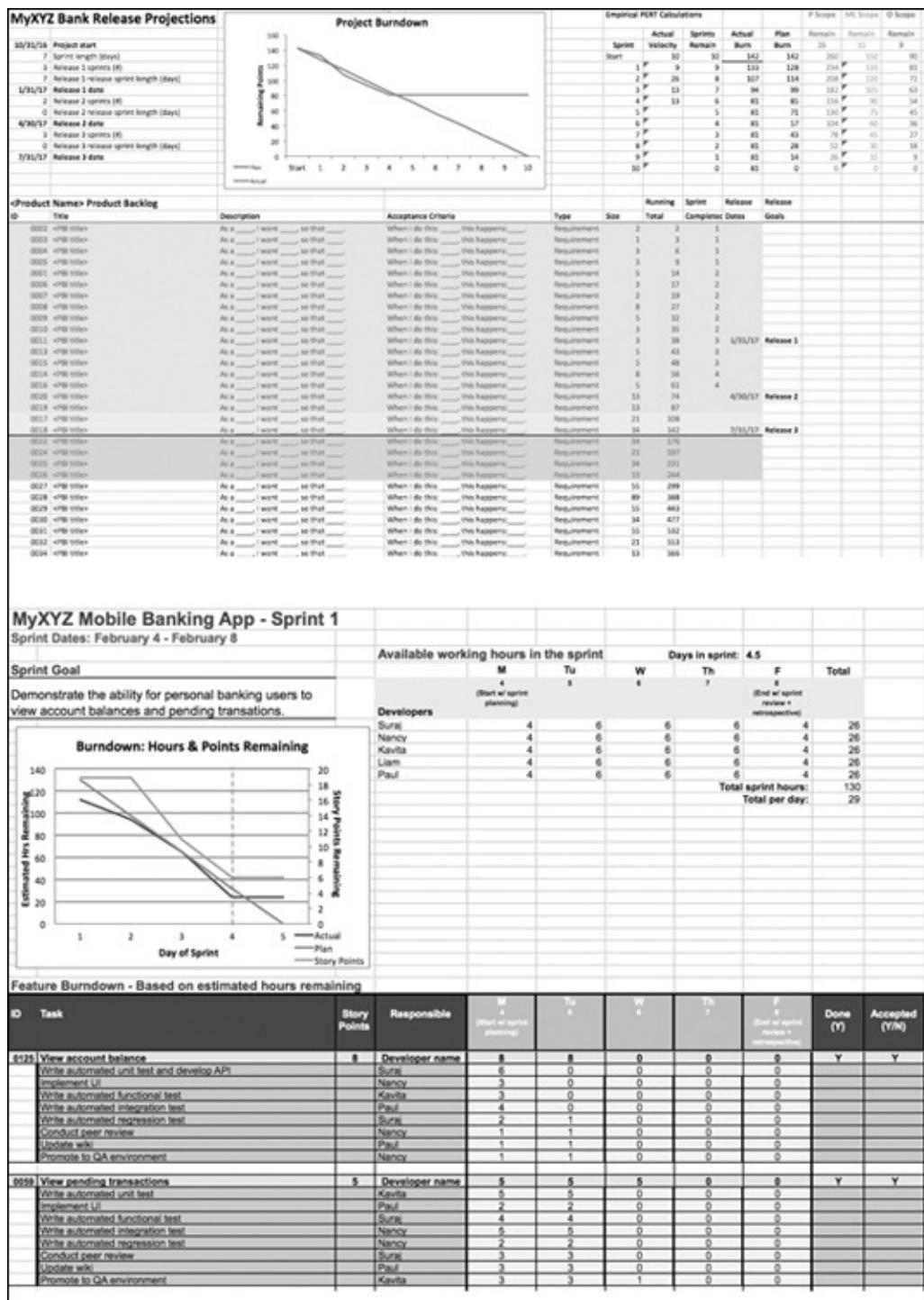
Our ability to define the features of a system increases exponentially when we step up our interaction with the proposed solution: A graphical representation is almost always better than a textual one, and experiencing functionality hands-on is best.



**TIP** Even a sketch on a piece of paper can be a more effective communication tool than a formal text-based document. A picture is worth a thousand words. A textual description is the weakest form of communication if you're trying to ensure common understanding — especially when it's delivered by email with the request to "let me know if you have any questions."

Examples of strategies for visualization include the following:

- » Stocking the work environment with plenty of whiteboards, poster paper, pens, and paper so that drawing tools are readily available
- » Using models instead of text to communicate concepts
- » Reporting project status through charts, graphs, and dashboards, such as those in [Figure 2-2](#).



**FIGURE 2-2:** Charts, graphs, and dashboards for reporting project status.

## Changes as a Result of Agile Values

The publication of the Agile Manifesto and the 12 Agile Principles legitimized and focused the agile movement in the following ways:

- » **Agile approaches changed attitudes toward project management processes.** In trying to improve processes, methodologists in the past worked to develop a universal process that could be used under all conditions, assuming that more process and greater formality would yield improved results. This approach, however, required more time, overhead, and cost and often diminished quality. The manifesto and the 12 principles acknowledged that too much process is a problem, not a solution, and that the right process in the right amount differs in each situation.
- » **Agile approaches changed attitudes toward knowledge workers.** IT groups began to remember that development team members aren't disposable resources but individuals whose skills, talents, and innovation make a difference to every project. The same product created by different team members will be a different product.
- » **Agile approaches changed the relationship between business and IT groups.** Agile project management addressed the problems associated with the historical separation between business and IT by bringing these contributors together on the same project team, at equal levels of involvement and with shared goals.
- » **Agile approaches corrected attitudes toward change.** Historical approaches viewed change as a problem to be avoided or minimized. The Agile Manifesto and its principles helped identify change as an opportunity to ensure that the most informed ideas were implemented.

## CHANGES TO COME

Enterprises are leveraging agile techniques on a large-scale basis to solve business problems. Although the methodologies of agile IT groups, as well as non-IT groups, have undergone radical transformation, the organizations around these groups have often continued to use historical methodologies and concepts. For example, corporate funding and spending cycles are still geared toward the following:

- Long development efforts that deliver working software at the end of the project
- Annual budgeting
- An assumption that certainty is possible at the beginning of a project
- Corporate incentive packages focused on individual rather than team performance

The resulting tension keeps organizations from taking full advantage of the efficiency and

significant savings that agile techniques promise.

A truly integrated agile approach encourages organizations to move away from yesterday's traditions and develop a structure at all levels that continually asks what's best for the customer, the product, and the project team.

An agile project team can be only as agile as the organization it serves. As the movement continues to evolve, the values articulated in the Agile Manifesto and its principles provide a strong foundation for the changes necessary to make individual projects and entire organizations more productive and profitable. This evolution will be driven by passionate practitioners who continue to explore and apply agile principles and practices.

## *The Agile Litmus Test*

To be agile, you need to be able to ask, "Is this agile?" If you're ever in doubt about whether a particular process, practice, tool, or approach adheres to the Agile Manifesto or the 12 principles, refer to the following list of questions:

1. Does what we're doing at this moment support the early and continuous delivery of valuable software?
2. Does our process welcome and take advantage of change?
3. Does our process lead to and support the delivery of working functionality?
4. Are the developers and the product owner working together daily? Are customers and business stakeholders working closely with the project team?
5. Does our environment give the development team the support it needs to get the job done?
6. Are we communicating face to face more than through phone and email?
7. Are we measuring progress by the amount of working functionality produced?
8. Can we maintain this pace indefinitely?
9. Do we support technical excellence and good design that allows for future changes?
10. Are we maximizing the amount of work not done — namely, doing as little as necessary to fulfill the goal of the project?

11. Is this development team self-organizing and self-managing? Does it have the freedom to succeed?
12. Are we reflecting at regular intervals and adjusting our behavior accordingly?

If you answered “yes” to all these questions, congratulations; you’re truly working on an agile project. If you answered “no” to any of these questions, what can you do to change that answer to “yes”? You can come back to this exercise at any time and use the agile litmus test with your project team and the wider organization.

# Chapter 3

## Why Being Agile Works Better

---

### IN THIS CHAPTER

- » Discovering the benefits of agile project management
- » Comparing agile approaches to historical approaches
- » Finding out why people like agile techniques

Agile approaches work well in the real world. Why is this? In this chapter, you examine the mechanics of how agile processes improve the way people work and how they prevent burdensome overhead. Comparisons with historical methods highlight the improvements agile techniques bring.

When talking about agile project management advantages, the bottom line is twofold: project success and stakeholder satisfaction.

### *Evaluating Agile Benefits*

The agile concept of project management is different from previous methodologies. As mentioned in [Chapter 1](#), agile approaches address key challenges of historical project management methods such as waterfall, but they also go much deeper. Agile processes provide a framework for how we *want* to work — how we naturally function when we solve problems and complete tasks.

Historical methods of project management were developed not for contemporary development lifecycles, such as software development, but for less complex systems. They also were adapted from other spheres, such as construction, manufacturing, and the military. It's no wonder that these project management methods don't fit when attempting to build more complex, modern products, such as mobile applications or web-centric, object-oriented applications, which require constant innovation to stay competitive. Even with older technologies, the track record of traditional

methodologies is abysmal, especially when applied to software projects. For more details on the high failure rates of projects that are run traditionally, check out the studies from the Standish Group shown in [Chapter 1](#).



**REMEMBER** You can use agile project management techniques in many industries besides software development. If you're creating a product and want early feedback throughout the process, you can benefit from agile processes.

When you have a critical looming deadline, your instinct is to *go agile*. Formality goes out of the window as you roll up your sleeves and focus on what has to get done. You solve problems quickly, practically, and in descending order of necessity, making sure you complete the most critical tasks.

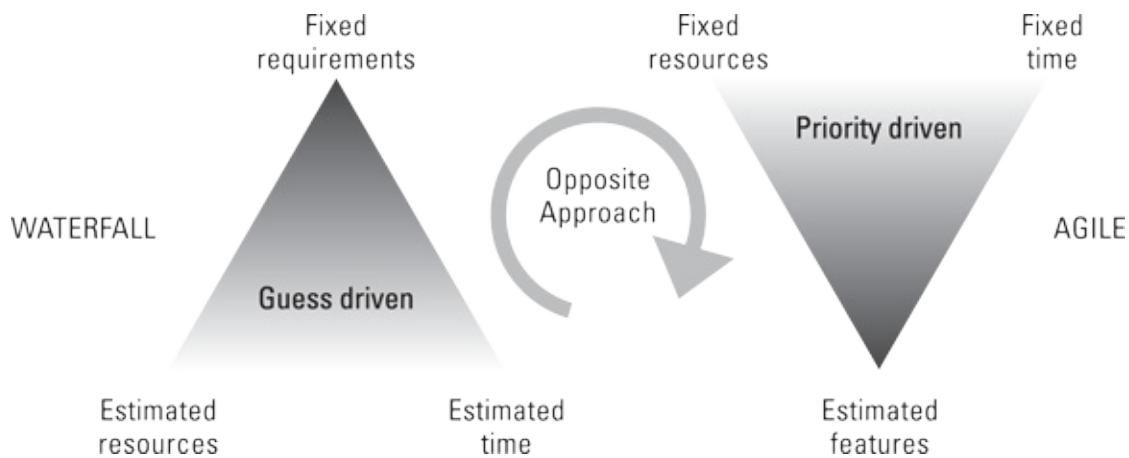
More than going agile — it's about *being agile*. When you become agile, you don't institute unreasonable deadlines to force greater focus. Instead, you realize that people function well as practical problem solvers, even under stress. For example, a popular team-building exercise titled the *marshmallow challenge* involves groups of four people building the tallest free-standing structure possible out of 20 sticks of spaghetti, a yard of tape, and a yard of string, and then placing a marshmallow on the top — in 18 minutes. See [www.marshmallowchallenge.com](http://www.marshmallowchallenge.com) for background information about the concept. On that site, you can also view the associated TED Talk by Tom Wujec.

Wujec points out that young children usually build taller and more interesting structures than most adults because children build incrementally on a series of successful structures in the time allotted. Adults spend a lot of time planning, produce one final version, and then run out of time to correct any mistakes. The youngsters provide a valuable lesson that *big bang development* — namely, excessive planning and then one shot at product creation — doesn't work. Formality, excessive time detailing uninformed future steps, and a single plan are often detriments to success.

The marshmallow challenge sets opening conditions that mimic those in real life. You build a structure (which equates to a software product in the IT

industry) using fixed resources (four people, spaghetti, and so on) and a fixed time (18 minutes). What you end up with is anyone's guess, but an underlying assumption in historical project management approaches is that you can determine the precise destination (the features or requirements) in the beginning and then estimate the people, resources, and time required.

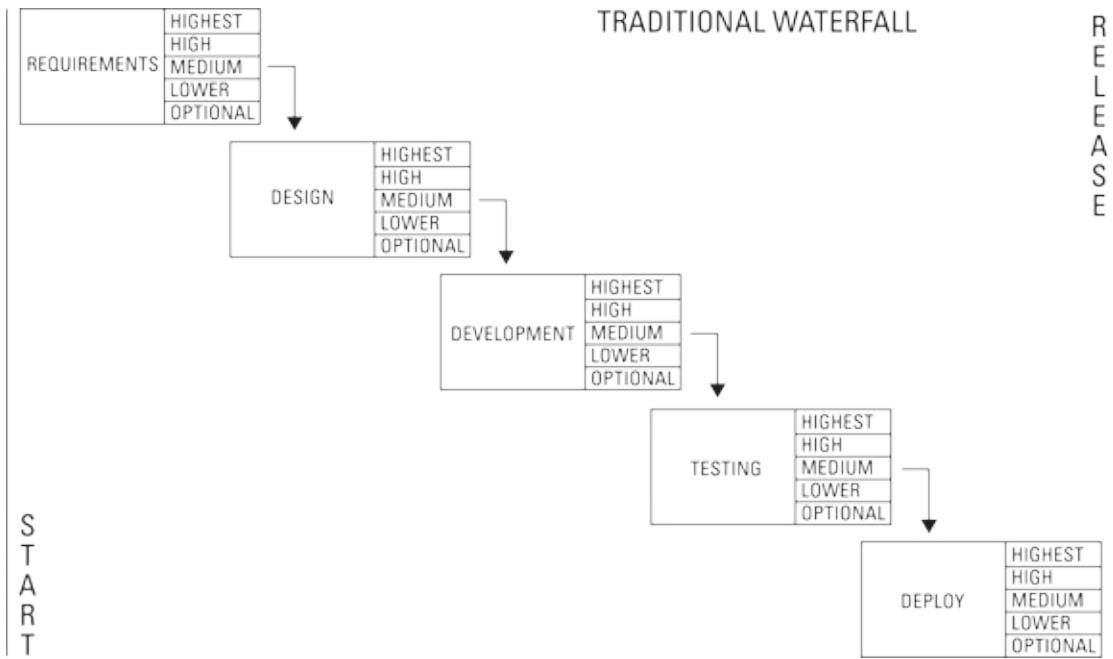
This assumption is upside down from how life really is. As you can see in [Figure 3-1](#), the theories of historical methods are the reverse of agile approaches. We pretend that we live in the world on the left, but we actually live in the world on the right.



[FIGURE 3-1:](#) A comparison of historical project management and agile concepts.

In the historical approach, which locks the requirements and delivers the product all in one go, the result is all or nothing. We either succeed completely or fail absolutely. The stakes are high because everything hinges on work that happens at the end (that is, putting the marshmallow on the top) of the final phase of the cycle, which includes integration and customer testing.

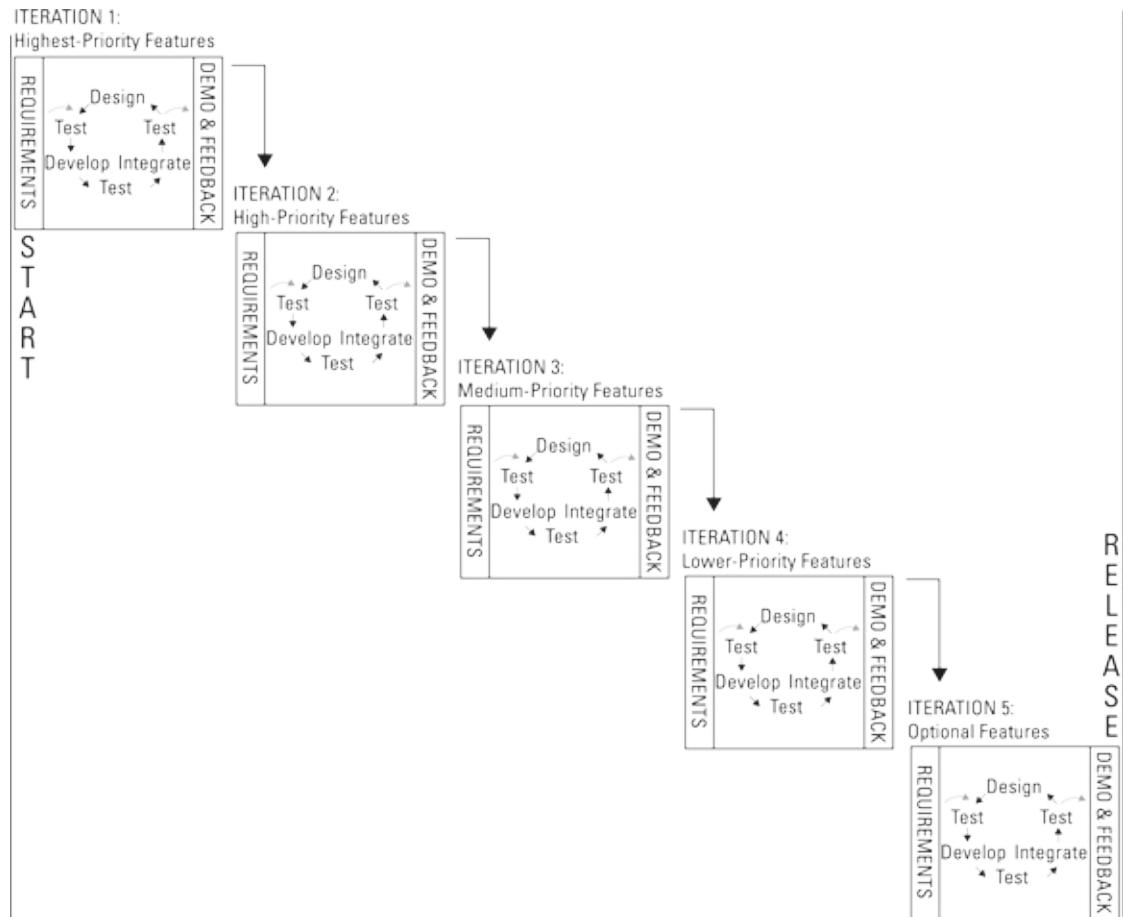
In [Figure 3-2](#), you can see how each phase of a waterfall project is dependent on the previous one. Teams design and develop all features together, meaning you don't get the highest-priority feature until you've finished developing the lowest-priority feature. The customer has to wait until the end of the project to get final delivery of any element of the product.



**FIGURE 3-2:** The waterfall project cycle is a linear methodology.

In the testing phase of a waterfall project, the customers get to see their long-awaited product. By that time, the investment and effort have been huge, and the risk of failure is high. Finding defects among all completed product requirements is like looking for a weed in a cornfield.

Agile project management turns the concept of how software development should be done upside down. Using agile methods, you develop, test, and launch small groups of product requirements in short iterative cycles, known as *iterations*, or *sprints*, as illustrated in [Figure 3-3](#). Testing occurs during each iteration. To find defects, the development team looks for a weed in a flowerpot, rather than in a cornfield.



**FIGURE 3-3:** Agile approaches have an iterative project cycle.

*Product owner, scrum master, and sprint* are terms from *scrum*, a popular agile framework for organizing work and exposing project progress. *Scrum* refers to a rugby huddle, in which a rugby team locks together over the ball. Scrum as an approach, like rugby, encourages the project team to work together closely and take responsibility for the result. (You find out more about scrum and other agile techniques in [Chapter 4](#).)

## WHERE THE WATERFALL FALLS SHORT

As we mention in [Chapter 1](#), before 2008, waterfall was the most widely used traditional project management methodology. The following list summarizes the major aspects of the waterfall approach to project management:

- The team must know all requirements up front to estimate time, budgets, team members, and resources. Knowing all the requirements at the project start means you have a high investment in detailed requirements gathering before any development begins.

- Estimation is complex and requires a high degree of competence and experience and a lot of effort to complete.
- The customer and stakeholders may not be available to answer questions during the development period, because they may assume that they provided all the information needed during the requirements-gathering and design phases.
- The team needs to resist the addition of new requirements or document them as change orders, which adds more work to the project and extends the schedule and budget.
- The team must create and maintain volumes of process documentation to manage and control the project.
- Although some testing can be done as you go, final testing can't be completed until the end of the project, when all functionality has been developed and integrated.
- Full and complete customer feedback is not possible until the end of the project, when all functionality is complete.
- Funding is ongoing, but the value appears only at the end of the project, creating a high level of risk.
- The project has to be fully complete for value to be achieved. If funding runs out prior to the end of the project, the project delivers zero value.

Moreover, on an agile project, the customers get to see their product at the end of every short cycle. You can create the highest-priority features first, which gives you the opportunity to ensure maximum value early on, when less of the customer's money has been invested.

This agile concept is attractive, especially to risk-averse organizations. In addition, if your product has market value, revenue can be coming in even during development. Now you have a self-funding project!

## ***How Agile Approaches Beat Historical Approaches***

Agile frameworks promise significant advantages over historical methods, including greater flexibility and stability, less nonproductive work, faster delivery with higher quality, improved development team performance, tighter project control, and faster failure detection. We describe all these results in this section.

However, these results can't be achieved without a highly competent and

functional development team. The development team is pivotal to the success of the project. Agile methods emphasize the importance of the support provided to the development team as well as the importance of project team members' actions and interactions.



**REMEMBER** The first core value in the Agile Manifesto is “Individuals and interactions over processes and tools.” Nurturing the development team is central to agile project management and the reason why you can have such success with agile approaches.

Agile project teams are centered on development teams (which include developers, testers, designers, and anyone else who does the actual work of creating the product), and also include project stakeholders, as well as the following two important team members, without which the development team couldn’t function:

- » **Product owner:** The *product owner* is a project team member who is an expert on the product and the customer’s business needs. The product owner works with the business community and prioritizes product requirements, and supports the development team by being available to provide daily clarifications and final acceptance to the development team. ([Chapter 2](#) has more on the product owner.)
- » **Scrum master or agile coach:** The *scrum master or agile coach* acts as a buffer between the development team and distractions that might slow down the development effort. The scrum master also provides expertise on agile processes and helps remove obstacles that hinder the development team from making progress. The scrum master or agile coach facilitates consensus building and stakeholder communication.

You can find complete descriptions of the product owner, the development team, and the scrum master in [Chapter 6](#). Later in this chapter, you see how the product owner and scrum master’s highest priority is supporting and optimizing the development team’s performance.

## ***Greater flexibility and stability***

By way of comparison, agile projects offer both greater flexibility and greater

stability than traditional projects. First, you find out how agile projects offer flexibility, and then we discuss stability.

A project team, regardless of its project management approach, faces two significant challenges at the beginning of a project:

- » The project team has limited knowledge of the product end state.
- » The project team cannot predict the future.

This limited knowledge of the product and of future business needs almost guarantees project changes.



**REMEMBER** The fourth core value in the Agile Manifesto is “Responding to change over following a plan.” The agile framework was created with flexibility in mind.

With agile approaches, project teams can adapt to new knowledge and new requirements that emerge as the project progresses. We provide many details about the agile processes that enable flexibility throughout this book. Here’s a simple description of some processes that help agile project teams manage change:

- » At the start of an agile project, the product owner gathers high-level product requirements from project stakeholders and prioritizes them. The product owner doesn’t need all the requirements — just enough to have a good understanding of what the product must accomplish.
- » The development team and the product owner work together to break down the initial highest-priority requirements into more detailed requirements. The result is small chunks of work that the development team can start developing immediately.
- » You focus on the top priorities in each sprint regardless of how soon before the sprint those priorities were set.

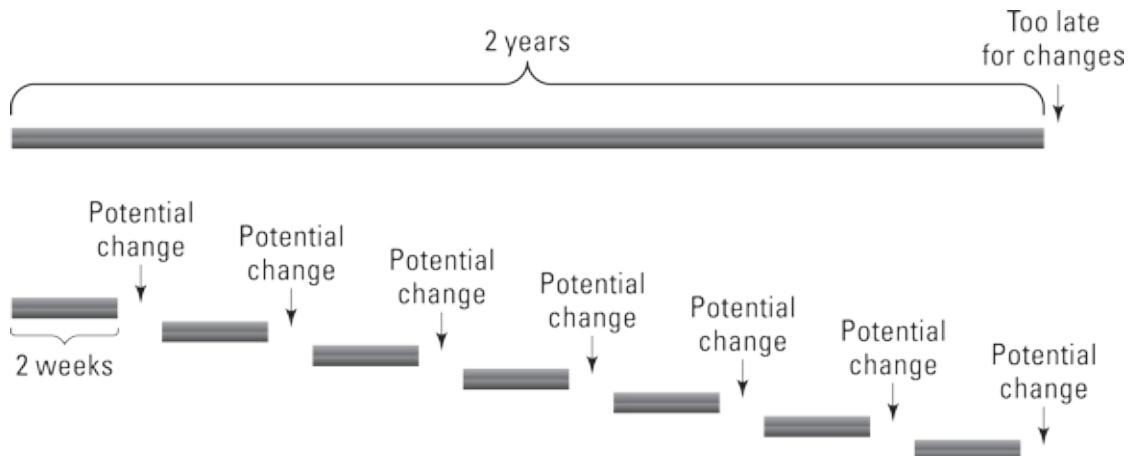


**TIP** Iterations, or sprints, on agile projects are short — they last up to

four weeks, and are often one or two weeks. You can find details about sprints in [Chapters 8–10](#).

- » The development team works on groups of requirements within sprints and learns more about the product with each successive sprint.
- » The development team plans one sprint at a time and drills further into requirements at the beginning of each sprint. The development team generally works only on the highest-priority requirements.
- » Concentrating on one sprint at a time and on the highest-priority requirements allows the project team to accommodate new high-priority requirements at the beginning of each sprint.
- » When changes arise, the product owner updates a list of requirements that remains to be dealt with in future sprints. The product owner reprioritizes the list regularly.
- » The product owner can financially invest in high-priority features first and can choose which features to fund throughout the project.
- » The product owner and development team collect client feedback at the end of each sprint and act on that feedback. Client feedback often leads to changes to existing functionality or to new, valuable requirements. Feedback can also lead to removing or reprioritizing requirements that are not really necessary.
- » The product owner can stop the project once he or she deems that the product has sufficient functionality to fulfill project goals.

[Figure 3-4](#) illustrates how making changes on agile projects can be more stable than making changes in waterfall. Think of the two images in the figure as steel bars. In the top image, the bar represents a two-year project. The bar's length makes it much easier to distort, bend, and break. Project changes can be thought of in the same way — long projects are structurally vulnerable to instability because the planning stage of a project is different than the execution, when reality sets in, and there is no natural point of give in a long project.



**FIGURE 3-4:** Stability in flexibility on agile projects.

Now look at the bottom image in [Figure 3-4](#). The small steel bars represent two-week iterations within a project. It is much easier for those small bars to be stable and unchanging than it is for the larger bar. In the same manner, it is easier to have project stability in smaller increments with known flexibility points. Telling a business there can be no changes for two weeks is much easier and more realistic than saying there can be no changes for two years.

Agile projects are tactically flexible because they are strategically stable. They're great at accommodating change because the means for regular change are built into everyday processes. At the same time, iterations on agile projects offer distinct areas for project stability. Agile project teams accommodate changes to the product backlog anytime but do not generally accommodate external changes to scope during the sprint. The product backlog may be constantly changing, but, except in emergencies, the sprint is generally stable.

At the beginning of the iteration, the development team plans the work it will complete for that sprint. After the sprint begins, the development team works only on the planned requirements. A couple of exceptions to this plan can occur — if the development team finishes early, it can request more work; if an emergency arises, the product owner can cancel the sprint. In general, however, the sprint is a time of great stability for the development team.

This stability can lead to innovation. When development team members have stability — that is, they know what they will be working on in a set period of time — they will think about their tasks consciously at work. They may also think about tasks unconsciously away from work and tend to come up with

solutions at any given time.

Agile projects provide a constant cycle of development, feedback, and change, allowing project teams the flexibility to create products with only the right features and the stability to be creative.

## ***Reduced nonproductive tasks***

When you're creating a product, at any point in your working day, you can work either on developing the product or on the peripheral processes that are supposed to manage and control the creation of the product. Clearly, there's more value in the first, which you should try to maximize, than in the second, which you want to minimize.

To finish a project, you have to work on the solution. As obvious as this statement sounds, it's routinely neglected on waterfall projects. Programmers on some software projects spend only 20 percent of their time generating functionality, with the rest of the time in meetings, writing emails, or creating unnecessary presentations and documentation.

Product development can be an intense activity that requires sustained periods of focus. Many developers can't get enough development time during their normal workday to keep up with the schedule of a project because they're doing other types of tasks. The following causal chain is the result:

Long workday = tired developers = unnecessary defects = more defect fixing = delayed release = longer time to value

To maximize productive work, the goal is to eliminate overtime and have developers creating functionality during the working day. To increase productive work, you have to reduce unproductive tasks, period.

## ***Meetings***

Meetings can be a large waste of valuable time. On traditional projects, development team members may find themselves in long meetings that provide little or no benefit to the developers. The following agile approaches can help ensure that development teams spend time only in productive, meaningful meetings:

- » Agile processes include only a few formal meetings. These meetings are focused, with specific topics and limited time. On agile projects, you

generally don't need to attend non-agile meetings.

- » Part of the scrum master's job is to prevent disruptions to the development team's working time, including requests for non-agile meetings. When there's a demand to pull developers away from development work, the scrum master asks "why" to understand the true need. The scrum master then may figure out how to satisfy that need without disrupting the development team.
- » On agile projects, the current project status is often visually available to the entire organization, removing the need for status meetings. You can find ways to streamline status reporting in [Chapter 14](#).

## Email

Email is not an efficient mode of communication; agile project teams aim to use email only sparingly. The email process is asynchronous and slow: You send an email, you wait for an answer; you have another question, you send another email. This process eats up time that could be spent more productively.

Instead of sending emails, agile project teams use face-to-face discussions to resolve questions and issues on the spot.

## Presentations

When preparing for a presentation of the functionality to the customer, agile project teams often use the following techniques:

- » **Demonstrate, don't present.** In other words, show the customer what you've created, rather than describing what you've created.
- » **Show how the functionality delivers on the requirement and fulfills the acceptance criteria.** In other words, say, "This was the requirement. These are the criteria needed to indicate that the feature was complete. Here is the resulting functionality meeting those criteria."
- » **Avoid formal slide presentations and all the preparation they involve.** When you demonstrate the working functionality, it will speak for itself. Keep demonstrations raw and real.

## Process documentation

Documentation has been the burden of project managers and developers for a

long time. Agile project teams can minimize documentation with the following approaches:

- » **Use iterative development.** A lot of documentation is created to reference decisions made months or years ago. Iterative development shortens the time between decision and developed product from months or years to days. The product and associated automated tests, rather than extensive paperwork, documents the decisions made.
- » **Remember that one size doesn't fit all.** You don't have to create the same documents for every project. Choose what you need to fit the particular project.
- » **Use informal, flexible documentation tools.** Whiteboards, sticky notes, charts, and other visual representations of the work plan are great tools.
- » **Include simple tools that provide adequate information for management about project progress.** Don't create special project progress reports, such as extensive status reports, for the sake of reporting. Agile teams use visual charts, such as burndown charts, to readily convey project status.

## ***Higher quality, delivered faster***

On traditional projects, the period from completion of requirements gathering to the beginning of customer testing can be painfully long. During this time, the customer is waiting to see some sort of result, and the development team is wrapped up in developing. The project manager is making sure that the project team is following the plan, keeping changes at bay, and updating everyone with an interest in the outcome by providing frequent and detailed reports.

When testing starts, near the end of the project, defects can cause budget increases, create schedule delays, and even kill a project. Testing is a project's largest unknown, and in traditional projects, it is an unknown carried until the end.

Agile project management is designed to deliver high-quality, shippable functionality quickly. Agile projects achieve better quality and quick delivery with the following:

- » The client reviews working functionality at the end of each sprint, and gives immediate feedback to the team for inspection and adaptation as soon as the next sprint.
- » Short development iterations (sprints) limit the number and complexity of features in development at any given time, making the finished work easier to test in each sprint. Only so much can be created in each sprint. Development teams break down features too complex for one sprint.
- » The development team builds and tests daily and maintains a working product throughout the project.
- » The product owner is involved throughout the day to answer questions and clarify misunderstandings quickly.
- » The development team is empowered and motivated and has a reasonable workday. Because the development team is not worn out, fewer defects occur.
- » Errors are detected quickly because developers test their work as it's completed. Extensive automated testing happens frequently, at least every night.
- » Modern software development tools allow many requirements to be written as test scripts, without the need for programming, which makes automated testing quicker.

## ***Improved team performance***

Central to agile project management is the experience of the project team members. Compared with traditional approaches such as waterfall, agile project teams get more environmental and organizational support, can spend more time focusing on their work, and can contribute to the continuous improvement of the process. To find out what these characteristics mean in practice, continue reading.

## ***Support for the team***

The development team's ability to deliver potentially shippable functionality is central to getting results with agile approaches and is achieved with the following support mechanisms:

- » A common agile practice is *collocation* — keeping the development team

and, ideally, the product owner together in one place and physically close to the customer. Collocation encourages collaboration and makes communication faster, clearer, and easier. You can get out of your seat, have a direct conversation, and eliminate any vagueness or uncertainty immediately.

- » The product owner can respond to development team questions and requests for clarification without delay, eliminating confusion and allowing work to proceed smoothly.
- » The scrum master removes impediments and ensures that the development team has everything it needs to focus and achieve maximum productivity.

## ***Focus***

Using agile processes, the development team can focus as much of its work time as possible on the development of the product. The following approaches help agile development teams focus:

- » Development team members are allocated 100 percent to one project, eliminating the time and focus lost by switching context among different projects.
- » Development team members know that their teammates will be fully available.
- » Developers focus on small units of functionality that are as independent as possible from other functionality. Every morning, the development team knows what it means to be successful that day.
- » The scrum master has an explicit responsibility to help protect the development team from organizational distractions.
- » The time the development team spends on coding and related productive activities increases because nonproductive work decreases.

## ***Continuous improvement***

An agile process isn't a mindless check-the-box approach. Different types of projects and different project teams are able to adapt around their specific situation, as you see in the discussion of sprint retrospectives in [Chapter 10](#). Here are some ways that agile project teams can continuously improve:

- » Iterative development makes continuous improvement possible because each new iteration involves a fresh start.
- » Because sprints happen over only a few weeks, project teams can incorporate process changes quickly.
- » A review process called the *retrospective* takes place at the end of each iteration and gives all agile team members a specific forum for identifying and planning actions for improvements.
- » The entire scrum team — product owner, development team members, and scrum master — reviews aspects of the work it feels might need improvement.
- » The scrum team applies the lessons it learns from the retrospective to the sprints that follow, which thus become more productive.

## **Tighter project control**

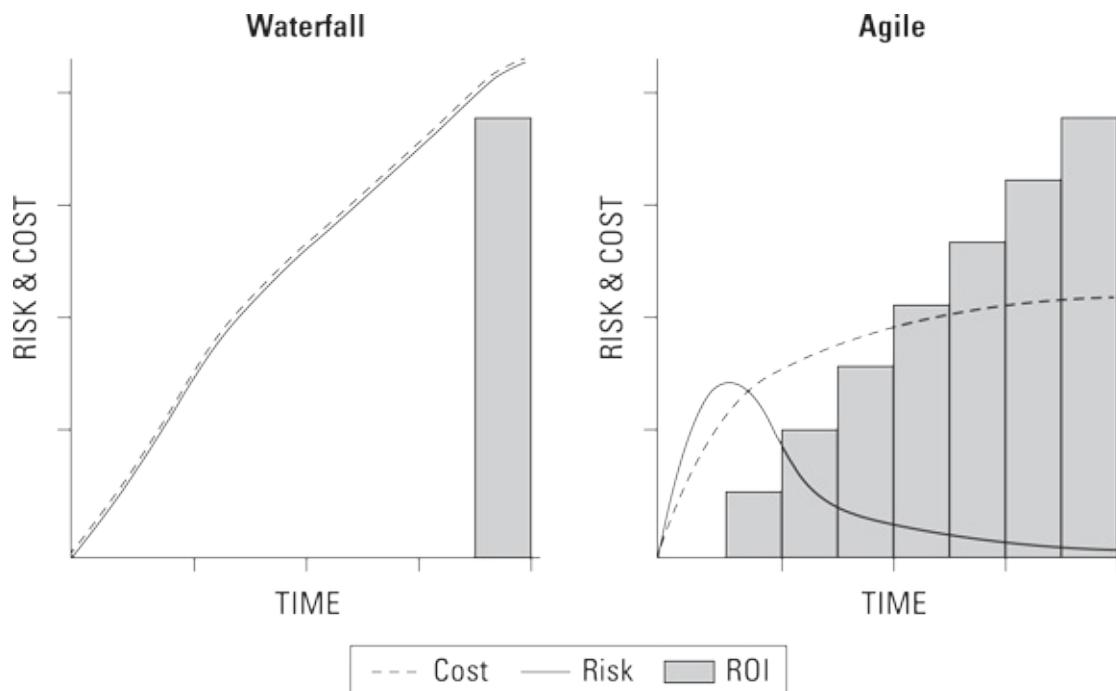
The work goes more quickly under agile projects than under waterfall conditions. Elevated productivity helps increase project control with the following:

- » Agile processes provide a constant flow of information. Development teams plan their work together every morning in daily scrum meetings, and they update task status throughout each day.
- » For every sprint, the customer has the opportunity to reprioritize product requirements based on business needs.
- » After you deliver working functionality at the end of each sprint, you finalize the workload for the next sprint according to current priorities. It makes no difference whether the priorities were set weeks or minutes before the next sprint.
- » When the product owner sets priorities for the next sprint, this action has no effect on the current sprint. On an agile project, a change in requirements adds no administrative costs or time and doesn't disrupt the current work.
- » Agile techniques make project termination easier. At the end of each iteration, you can determine whether the features of the product are now adequate. Low-priority items may never be developed.

In waterfall, project metrics may be outdated by weeks, and demonstrable functionality may be months away. In an agile context, metrics are fresh and relevant every day, work completed is often compiled and integrated daily, and working software is demonstrated every few weeks. From the first sprint to the close of the project, every project team member knows whether the project team is delivering. Up-to-the-minute project knowledge and the ability to quickly prioritize make high levels of project control possible.

## ***Faster and less costly failure***

In a waterfall project, opportunities for failure detection are theoretical until close to the end of the project schedule, when all the completed work comes together and when most of the investment is gone. Waiting until the final weeks or days of the project to find out that the product has serious issues is risky for all concerned. [Figure 3-5](#) compares the risk and investment profile for waterfall with that for agile approaches.



[FIGURE 3-5:](#) A risk and investment chart comparing waterfall and agile methodologies.

Along with opportunities for tighter project control, the agile framework offers you

- » Earlier and more frequent opportunities to detect failure
- » An assessment and action opportunity every few weeks

- » Reduction in failure costs

What sorts of failures have you seen on projects? Would agile approaches have helped? You can find out more about risk on agile projects in [Chapter 15](#).

## **Why People Like Being Agile**

You've seen how an organization can benefit from agile project management with faster product delivery and lower costs. In the following sections, you find out how the people involved in a project can benefit as well, whether directly or indirectly.

### **Executives**

Agile project management provides two benefits that are especially attractive to executives: efficiency and a higher and quicker return on investment.

### **Efficiency**

Agile practices allow for vastly increased efficiency in the development process in the following ways:

- » Agile development teams are very productive. They organize the work themselves, focus on development activities, and are protected from distractions by the scrum master.
- » Nonproductive efforts are minimized. The agile approach eliminates unfruitful work; the focus is on development.
- » By using simple, timely, on-demand visual aids — such as graphs and diagrams — to display what's been done, what's in progress, and what's to come, the progress of the project is easier to understand at a glance.
- » Through continuous testing, defects are detected and corrected early.
- » An agile project can be halted when it has enough functionality.

### **Increased ROI opportunity**

ROI is significantly enhanced using agile approaches for the following reasons:

- » **Functionality is delivered to the marketplace earlier.** Features are fully

completed and then released in groups, rather than waiting until the end of all development and releasing 100 percent of the features at once.

- » **Product quality is higher.** The scope of development is broken down into manageable chunks that are tested and verified on an ongoing basis.
- » **Revenue opportunity can be accelerated.** Increments of the product are released to the market earlier than with traditional approaches to project management.
- » **Projects can self-fund.** A release of functionality might generate revenue while development of further features is ongoing.

## ***Product development and customers***

Customers like agile projects because they can accommodate changing requirements and generate higher-value products.

### ***Improved adaptation to change***

Changes to product requirements, priorities, timelines, and budgets can greatly disrupt traditional projects. In contrast, agile processes handle project and product changes in beneficial ways. For example:

- » Agile projects create an opportunity for increased customer satisfaction and return on investment by handling change effectively.
- » Changes can be incorporated into subsequent iterations routinely and smoothly.
- » Because the team members and the sprint length remain constant, project changes pose fewer problems than with traditional approaches. Necessary changes are slotted into the features list based on priority, pushing lower-priority items down the list. Ultimately, the product owner chooses when the project will end, at the point where future investment won't provide enough value.
- » Because the development team develops the highest-value items first and the product owner controls the prioritization, the product owner can be confident that business priorities are aligned with developer activity.

### ***Greater value***

With iterative development, product features can be released as the

development team completes them. Iterative development and releases provide greater value in the following ways:

- » Project teams deliver highest-priority product features earlier.
- » Project teams can deliver valuable products earlier.
- » Project teams can adjust requirements based on market changes and customer feedback.

## ***Management***

People in management tend to like agile projects for the higher quality of the product, the decreased waste of time and effort, and the emphasis on the value of the product over checking off lists of features of dubious usefulness.

### ***Higher quality***

With software development, through such techniques as test-driven development, continuous integration, and frequent customer feedback on working software, you can build higher quality into the product upfront.

With non-software development projects, what are ways you can think of to build in quality upfront?

### ***Less product and process waste***

In agile projects, wasted time and features are reduced through a number of strategies, including the following:

- » **Just-in-time (JIT) elaboration:** Amplification of only the currently highest-priority requirements means that time isn't spent working on details for features that might never be developed.
- » **Customer and stakeholder participation:** Customers and other stakeholders can provide feedback in each sprint, and the development team incorporates that feedback into the project. As the project and feedback continue, value to the customer increases.
- » **A bias for face-to-face conversation:** Faster, clearer communication saves time and confusion.
- » **Built-in exploitation of change:** Only high-priority features and functions are developed.

- » **Emphasis on the evidence of working functionality:** If a feature doesn't work or doesn't work in a valuable way, it's discovered early at a lower cost.

### ***Emphasis on value***

The agile principle of simplicity supports the elimination of processes and tools that don't support development directly and efficiently, and the exclusion of features that add little tangible value. This principle applies to administration and documentation as well as development in the following ways:

- » Fewer, shorter, more focused meetings
- » Reduction in pageantry
- » Barely sufficient documentation
- » Joint responsibility between customer and project team for the quality and value of the product

### ***Development teams***

Agile approaches empower development teams to produce their best work under reasonable conditions. Agile methods give development teams

- » A clear definition of success through joint sprint goal creation and identification of the acceptance criteria during requirements development
- » The power and respect to organize development as they see fit
- » The customer feedback they need to provide value
- » The protection of a dedicated scrum master to remove impediments and prevent disruptions
- » A humane, sustainable pace of work
- » A culture of learning that supports both personal development and project improvement
- » A structure that minimizes non-development time

Under the preceding conditions, the development team thrives and delivers results faster and with higher quality.



**REMEMBER** On Broadway and in Hollywood, performers who are on stage and onscreen to connect with the audience are often referred to as “the talent.” They are the reason many entertainment customers come to a show, and the supporting writers, directors, and producers ensure that they shine. In an agile environment, the development team is “the talent.” When the talent is successful, everyone succeeds.

## Part 2

# Being Agile

## **IN THIS PART ...**

Understand what it means to be agile and how to put agile practices into action.

Get an overview of the three most popular agile approaches, and discover how to create the right environment of physical space, communication, and tools to facilitate agile interactions.

Examine the behavior shift in values, philosophy, roles, and skills needed to operate in an agile team.

# Chapter 4

# Agile Approaches

---

## IN THIS CHAPTER

- » Applying agile practices
- » Understanding lean, scrum, and extreme programming
- » Connecting agile techniques

In previous chapters, you read about the history of agile project management. You may have even heard of common agile frameworks and techniques. Are you wondering what agile frameworks, methods, and techniques actually look like? In this chapter, you get an overview of three of the most common approaches used today to implement an agile project.

## *Diving under the Umbrella of Agile Approaches*

The Agile Manifesto and the agile principles on their own wouldn't be enough to launch you into an agile project, eager as you might be to do so. The reason is that principles and practices are different. The approaches described in this book, however, provide you with the necessary practices to be successful on an agile project.

*Agile* is a descriptive term for a number of techniques and methods that have the following similarities:

- » Development within multiple iterations, called *iterative development*
- » Emphasis on simplicity, transparency, and situation-specific strategies
- » Cross-functional, self-organizing teams
- » Working functionality as the measure of progress

Agile project management is an empirical project management approach. In other words, you do something in practice and adjust your approach based on experience rather than theory.

With regards to product development, the empirical approach is braced by the following pillars:

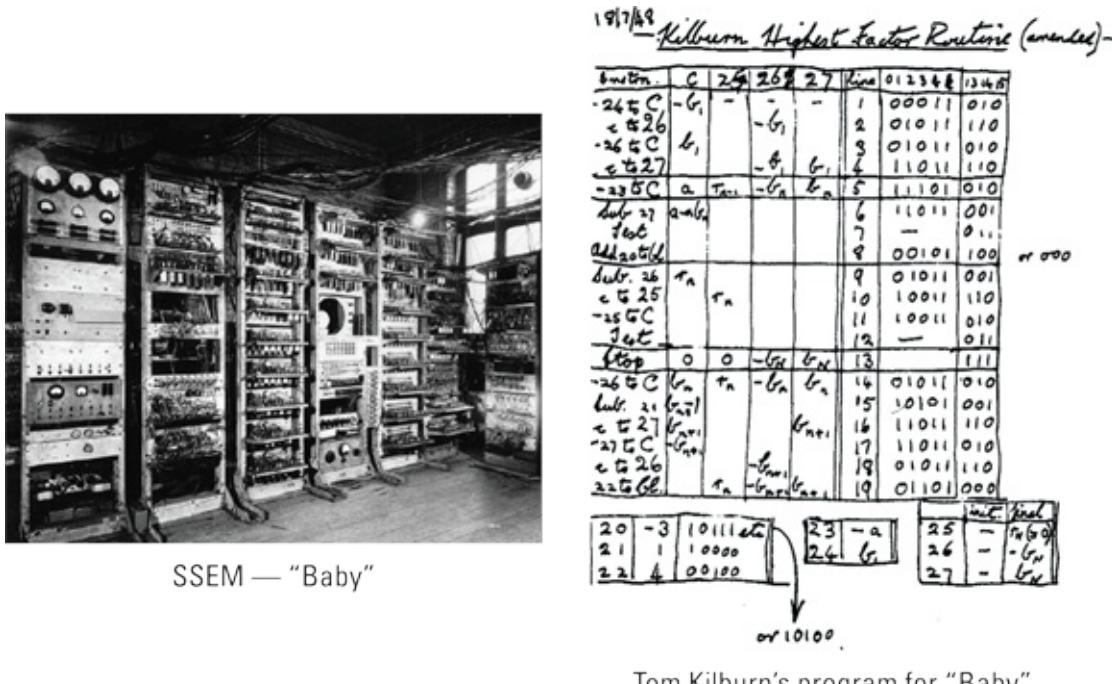
- » **Unfettered transparency:** Everyone involved in the process understands and can contribute to the development of the process.
- » **Frequent inspection:** The inspector must inspect the product regularly and possess the skills to identify variances from acceptance criteria.
- » **Immediate adaptation:** The development team must be able to adjust quickly to minimize further product deviations.

A host of approaches have agile characteristics. Three, however, are common to many agile projects: lean product development, scrum, and extreme programming (XP). These three approaches work perfectly together and share many common elements, although they use different terminology or have a slightly different focus. Broadly, lean and scrum focus on structure. Extreme programming does that, too, but is more prescriptive about development practices, focusing more on technical design, coding, testing, and integration. (From an approach called *extreme programming*, this type of focus is to be expected.) When organizations we work with state that they're using an agile approach for managing projects, they're usually working in an environment that is lean, with constant attention to limiting work in progress, wasteful practices, and process steps; using scrum to organize their work and expose project progress; and using extreme programming practices to build in quality upfront. Each of these approaches is explained in more detail later in this chapter.

Like any systematic approach, agile techniques didn't arise out of nothing. The concepts have historical precedents, some of which have origins outside software development, which isn't surprising, given that software development hasn't been around that long in the history of human events.

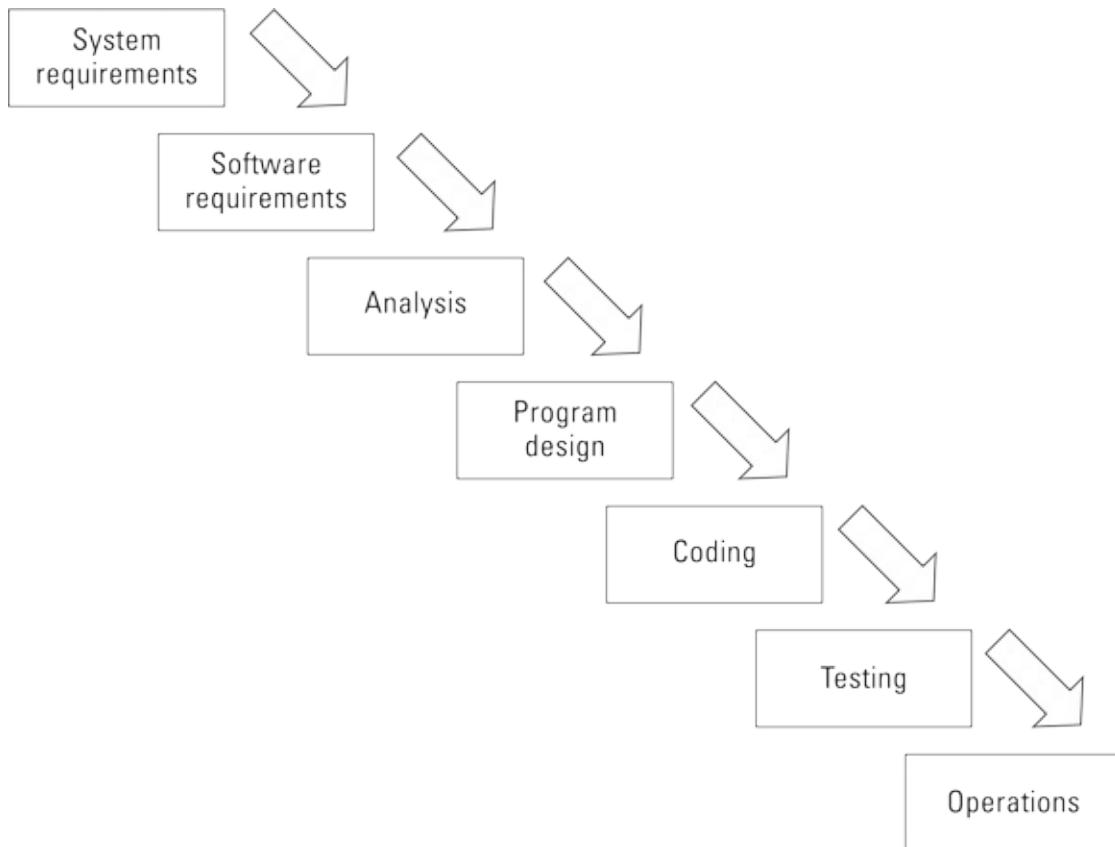
The basis for agile approaches is not the same as that of traditional project management methodologies such as waterfall, which was rooted in a defined control method used for World War II materials procurement. Early computer

hardware pioneers used the waterfall process to manage the complexity of the first computer systems, which were mostly hardware: 1,600 vacuum tubes but only 30 or so lines of hand-coded software. (See [Figure 4-1](#).) An inflexible process is effective when the problems are simple and the marketplace is static, but today's product development environment is too complex for such an outdated model.



**FIGURE 4-1:** Early hardware and software

Enter Dr. Winston Royce. In his article, “Managing the Development of Large Systems,” published in 1970, Dr. Royce codified the step-by-step software development process known as waterfall. When you look at his original diagram in [Figure 4-2](#), you can see where that name came from.



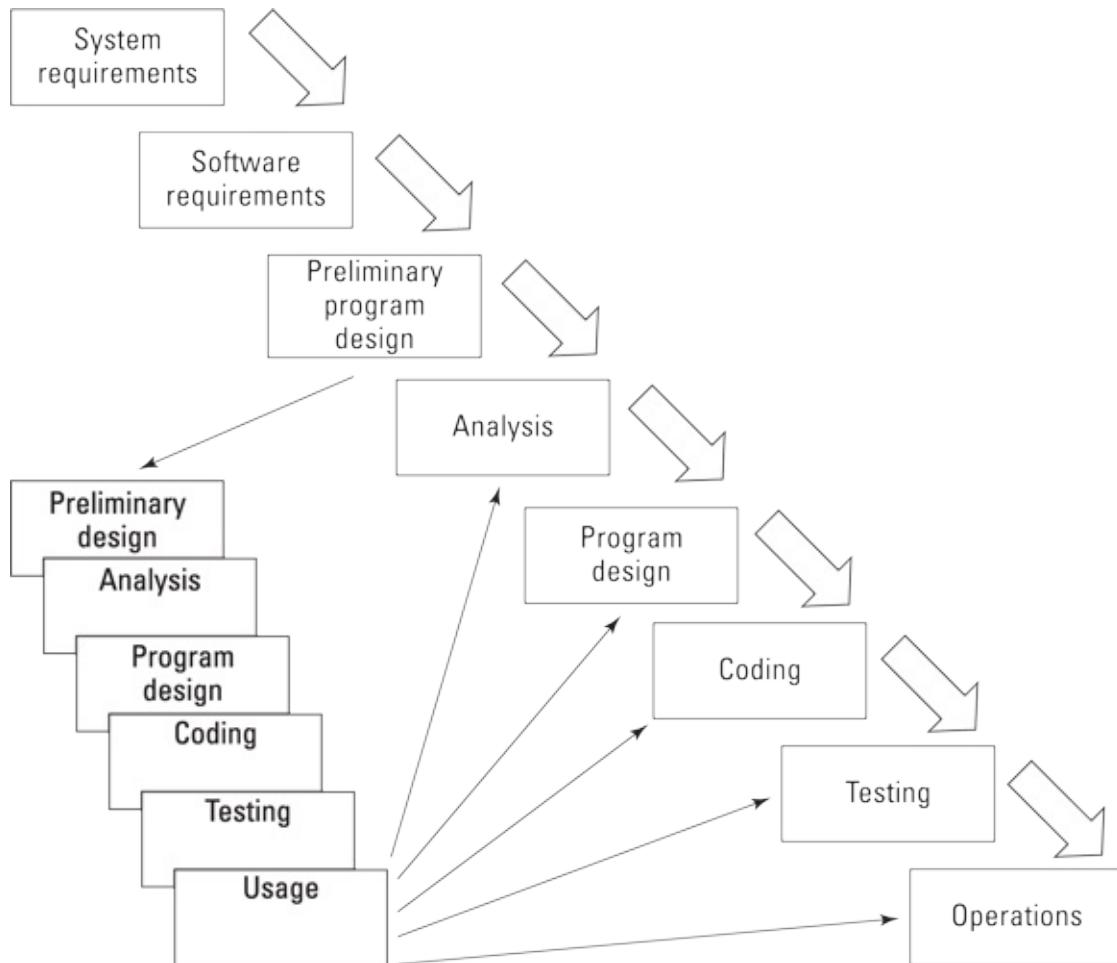
**FIGURE 4-2:** The origins of waterfall.

Over time, however, the computer development situation reversed. Hardware became repeatable through mass production, and software became the more complex and diverse aspect of a complete solution.

The irony here is that, even though the diagram implies that you complete tasks step by step, Dr. Royce himself added the cautionary note that you need iteration. Here's how he stated it:

“If the computer program in question is being developed for the first time, arrange matters so that the version being delivered to the customer for operational deployment is actually the second version insofar as critical design/operations areas are concerned.”

Royce even included the diagram shown in [Figure 4-3](#) to illustrate that iteration.



**FIGURE 4-3:** Iteration in waterfall.

Now, we're not sure if the diagram was stuck with chewing gum to other pages, but the software development community by and large lost this part of the story. After you allow the idea that you might not know everything when you first start developing a software component and might have to revisit the code to ensure that it's appropriate, you have the ray of light that lets in agile concepts. Agile might have come to prominence 40 years earlier if people had taken Dr. Royce's advice to heart!

## ***Reviewing the Big Three: Lean, Scrum, and Extreme Programming***

Now that you have a brief history of the waterfall approach to project management, you're ready to find out more about three popular agile

approaches: lean, scrum, and extreme programming.

## **An overview of lean**

Lean has its origins in manufacturing. Mass production methods, which have been around for more than 100 years, were designed to simplify assembly processes (for example, putting together a Model-T Ford). These processes use complex, expensive machinery and low-skilled workers to inexpensively churn out an item of value. The idea is that if you keep the machines and people working and stockpile inventory, you generate a lot of efficiency.

The simplicity is deceptive. Traditionally, mass production requires wasteful supporting systems and large amounts of indirect labor to ensure that manufacturing continues without pause. It generates a huge inventory of parts, extra workers, extra space, and complex processes that don't add direct value to the product. Sound familiar?

## **Cutting the fat as lean emerges in manufacturing**

In the 1940s in Japan, a small company called Toyota wanted to produce cars for the Japanese market but couldn't afford the huge investment that mass production requires. The company studied supermarkets, noting how consumers buy just what they need because they know there will always be a supply and how the stores restock shelves only as they empty. From this observation, Toyota created a just-in-time process that it could translate to the factory floor.

The result was a significant reduction in inventory of parts and finished goods and a lower investment in machines, people, and space.

One big cost of the mass production processes at the time was that humans on the production line were treated like machines: People had no autonomy and could not solve problems, make choices, or improve processes. The work was boring and set aside human potential. By contrast, the just-in-time process gives workers the ability to make decisions about what is most important to do next, in real time, on the factory floor. The workers take responsibility for the results. Toyota's success with just-in-time processes has helped change mass manufacturing approaches globally.

## **Understanding lean and software development**

The term *lean* was coined in the 1990s in *The Machine That Changed the World: The Story of Lean Production* (Free Press) by James P. Womack,

Daniel T. Jones, and Daniel Roos. eBay was an early adopter of lean principles for software development. The company led the way with an approach that responded daily to customers' requests for changes to the website, developing high-value features in a short time.

The focus of lean is business value and minimizing activities outside product development. Mary and Tom Poppendieck discuss a group of lean principles on their blog and in their books on lean software development. Following are the lean principles from their 2003 book, *Lean Software Development* (Addison-Wesley Professional):

- » **Eliminate waste.** Doing anything that is beyond barely sufficient (process steps, artifacts, meetings) slows down the flow of progress. Waste includes failing to learn from work, building the wrong thing, and thrashing (context switching between tasks or projects) — which results in only partially creating lots of product features but not completely creating any.
- » **Amplify learning.** Learning drives predictability. Enable improvement through a mindset of regular and disciplined transparency, inspection, and adaptation. Encourage an organization-wide culture that allows failure for the sake of learning from it.
- » **Deliver as late as possible.** Allow for late adaptation. Don't deliver late, but leave your options open long enough to make decisions at the last responsible moment based on facts rather than uncertainty — when you know the most. Learn from failure. Challenge standards. Use the scientific method — experiment with hypotheses to find solutions.
- » **Deliver as fast as possible.** Speed, cost, and quality are compatible. The sooner you deliver, the sooner you receive feedback. Work on fewer things at once, limiting work in progress and optimizing flow. Manage workflow, rather than schedules. Use just-in-time planning to shorten development and release cycles.
- » **Empower the team.** Working autonomously, mastering skills, and believing in the purpose of work can motivate development teams. Managers do not tell developers how to do their jobs but instead support them to self-organize around the work to be done and remove their impediments. Make sure teams and individuals have the environment and

tools they need to do their job well.

- » **Build quality in.** Establish mechanisms to catch and correct defects when they happen and before final verification. Quality is built in from the beginning, not at the end. Break dependencies, so you can develop and integrate functionality at any time without regressions.
- » **See the whole.** An entire system is only as strong as its weakest link. Solve problems, not just symptoms. Continually pay attention to bottlenecks throughout the flow of work and remove them. Think long-term when creating solutions.

Beyond the lean principles, one of the most common lean approaches used by agile teams is kanban, sometimes referred to as *lean-kanban*. Adapted from the Toyota Production System approach, *kanban* is essentially a method for removing waste to improve flow and throughput in a system.

Kanban practices can be applied in almost any situation because they're designed to start with where you are — you don't have to change anything about your existing workflow to get started. Kanban practices include the following:

- » Visualize.
- » Limit work in progress (WIP).
- » Manage flow.
- » Make process policies explicit.
- » Implement feedback loops.
- » Improve collaboratively, evolve experimentally.

The last three practices are commonly found in other agile frameworks, such as scrum and XP (both discussed later in this chapter). The first three enhance effectiveness for agile teams:

- » **Visualize:** Visualizing a team's workflow is the first step in identifying potential waste. Traditional bloated processes exist in many organizations but do not reflect reality, even if visualized. As agile teams visualize the flow of their work (on a whiteboard, on a wall, or in a drawing) and identify where productivity breaks down, they can easily analyze the root

cause and see how to remove the constraint. And then do it again, and again, and again.



TECHNICAL STUFF *Kanban* is Japanese for *visual signal*. Hanging on the factory wall or the development workspace wall, where everyone can see it, the kanban board shows the items that teams need to produce next. Slotted into the board are cards representing units of production. As production progresses, the workers remove, add, and move cards. As the cards move, they act as a signal to workers when work or inventory replenishment is needed. Agile teams use kanban boards or task boards to expose their progress and manage their flow of work (described in more detail in [Chapters 5 and 9](#)).

- » **Limit work in progress (WIP):** When teams keep starting work but don't finish it, their work in progress continues to grow. Being agile is all about getting to done, so the goal is to start things only when other things are completed. Working on multiple things at once does not mean you complete them all faster — you actually complete them more slowly than if you had worked on them one at a time. When agile teams limit their work in progress, items get completed faster, speeding the pace of completing each item in their queue.
- » **Manage flow:** We've all experienced what happens on a busy street during rush hour. When there are more cars than the lanes of traffic can handle, all cars move more slowly. Everyone wants to get somewhere at the same time, and so everyone has to wait longer to get there. To manage flow better, we need to regulate the entry of vehicles into the flow of traffic or increase the number of lanes of traffic where congestion is highest. Like cars in traffic, development work items move more slowly if developers try to take them all on at once. Working on one thing at a time and identifying and removing constraints increases the flow of all items through the system.



TECHNICAL STUFF Measuring lead and cycle times helps agile teams monitor their

management of flow. A team determines the lead time by tracking the amount of time it takes a request for functionality to go from arrival in the queue to completed. They know the cycle time by tracking the time from when work begins to when it is completed. And the agile team optimizes flow by identifying and removing bottlenecks that keep its lead and cycle times from decreasing.



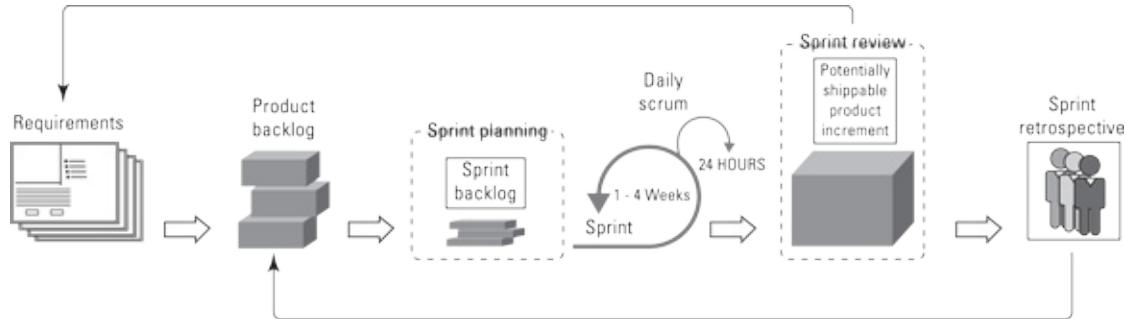
**REMEMBER** To support good product development practices, remember the following:

- » Don't develop features that you're unlikely to use.
- » Make the development team central to the project because it adds the biggest value.
- » Have the customers prioritize features — they know what's most important to them. Tackle high-priority items first to deliver value.
- » Use tools that support great communication across all parties.

Today, lean principles continue to influence the development of agile techniques — and to be influenced by them. Any approach should be agile and adapt over time.

## **An overview of scrum**

Scrum, the most popular agile framework in software development, is an iterative approach that has at its core the *sprint* (the scrum term for iteration). To support this process, scrum teams use specific roles, artifacts, and events. To make sure that they meet the goals of each part of the process, scrum teams use inspection and adaptation throughout the project. The scrum approach is shown in [Figure 4-4](#).



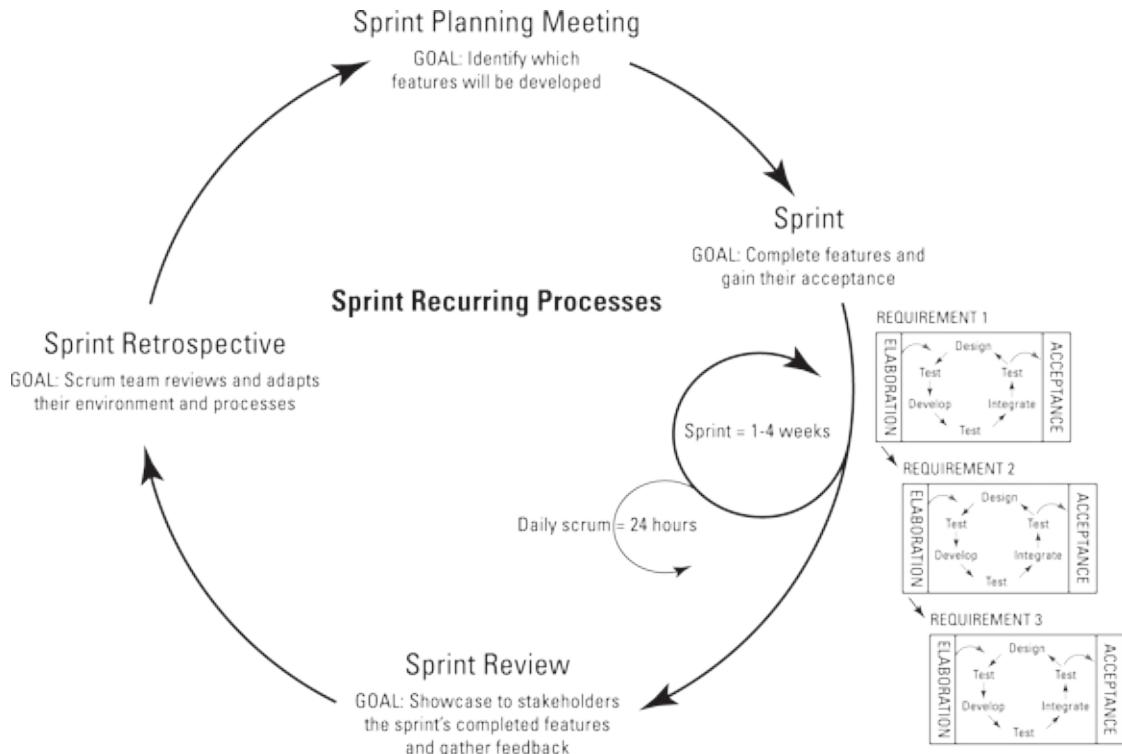
**FIGURE 4-4:** The scrum approach.

### *Going the distance with the sprint*

Within each sprint, the development team develops and tests a functional part of the product until the product owner accepts it, often daily, and the functionality becomes a potentially shippable increment of the overall product. When one sprint finishes, another sprint starts. Releasing functionality to the market often occurs at the end of multiple sprints, when the product owner determines that enough value exists. However, the product owner may decide to release functionality after every sprint, or even as many times as needed during a sprint.



**REMEMBER** A core principle of the sprint is its cyclical nature: The sprint, as well as the processes within it, repeats over and over, as shown in [Figure 4-5](#).



**FIGURE 4-5:** Sprints are recurring processes.

You use the tenets of inspection and adaptation on a daily basis as part of a scrum project:

- » During a sprint, you conduct *constant inspections* to assess progress toward the sprint goal, and consequentially, toward the release goal.
- » You hold a *daily scrum* meeting to organize the day by reviewing what the team completed yesterday and coordinating what it will work on today. Essentially, the scrum team inspects its progress toward the sprint goal.
- » At the end of the sprint, you use a *sprint retrospective* meeting to assess performance and plan necessary adaptations.

These inspections and adaptations may sound formal and process-laden, but they aren't. Use inspection and adaptation to solve issues and don't overthink this process. The problem you're trying to solve today will often change in the future anyway.

## ***Understanding scrum roles, artifacts, and events***

The scrum framework defines specific roles, artifacts, and events for projects.

Scrum's three *roles* — people on the project — are as follows:

- » **Product owner:** Represents and speaks for the business needs of the project.
- » **Development team:** Performs the day-to-day work. The development team is dedicated to the project and each team member is *multi-skilled* — that is, although team members may have certain strengths, each member is capable of doing multiple jobs on the project.
- » **Scrum master:** Protects the team from organizational distractions, clears roadblocks, ensures that scrum is played properly, and continuously improves the team's environment.

Additionally, scrum teams find that they're more effective and efficient when they work closely with two non-scrum-specific roles:

- » **Stakeholders:** Anyone who is affected by or has input on the project. Although stakeholders are not official scrum roles, it is essential for scrum teams and stakeholders to work closely together throughout a project.
- » **Agile mentor:** An experienced authority on agile techniques and the scrum framework. Often this person is external to the project's department or organization, so he or she can support the scrum team objectively with an outsider's point of view.

In the same way that scrum has specific roles, scrum also has three tangible deliverables, called *artifacts*:

- » **Product backlog:** The full list of requirements that defines the product, often documented in terms of business value from the perspective of the end user. The product backlog can be fluid throughout the project. All scope items, regardless of level of detail, are in the product backlog. The product owner owns the product backlog, determining what goes in it and in what priority.
- » **Sprint backlog:** The list of requirements and tasks in a given sprint. The product owner and the development team select the requirements for the sprint in sprint planning, with the development team breaking down these requirements into tasks. Unlike the product backlog, the sprint backlog

can be changed only by the development team.

- » **Product increment:** The usable, potentially shippable functionality. Whether the product is a website or a new house, the product increment should be complete enough to demonstrate its working functionality. A scrum project is complete after a product contains enough shippable functionality to meet the customer's business goals for the project.

Finally, scrum also has five events:

- » **Sprint:** Scrum's term for iteration. The *sprint* is the container for each of the other scrum events, in which the scrum team creates potentially shippable functionality. Sprints are short cycles, no longer than a month, typically between one and two weeks, and in some cases as short as one day. Consistent sprint length reduces variance; a scrum team can confidently extrapolate what it can do in each sprint based on what it has accomplished in previous sprints. Sprints give scrum teams the opportunity to make adjustments for continuous improvement immediately, rather than at the end of the project.
- » **Sprint planning:** Takes place at the start of each sprint. In sprint planning meetings, scrum teams decide which goal, scope, and supporting tasks will be part of the sprint backlog.
- » **Daily scrum:** Takes place daily for no more than 15 minutes. During the daily scrum, development team members make three statements:
  - What the team member completed yesterday
  - What the team member will work on today
  - A list of items impeding the team memberThe scrum master also participates in the context of impediments he or she is working to remove for the developers.
- » **Sprint review:** Takes place at the end of each sprint. In this meeting, the development team demonstrates to the stakeholders and the entire organization the accepted parts of the product the team completed during the sprint. The key to the sprint review is collecting feedback from the stakeholders, which informs the product owner how to update the product backlog and consider the next sprint goal.

» **Sprint retrospective:** Takes place at the end of each sprint. The sprint retrospective is an internal team meeting in which the scrum team members (product owner, development team, and scrum master) discuss what went well during the sprint, what didn't work well, and how they can make improvements for the next sprint. This meeting is action-oriented (frustrations should be vented elsewhere) and ends with tangible improvement plans for the next sprint.

Scrum is simple: three roles, three artifacts, and five events. Each plays a part to ensure that the scrum team has continuous transparency, inspection, and adaptation throughout the project. As a framework, scrum accommodates many other agile techniques, methods, and tools for executing the technical aspects of building functionality.

## ESSENTIAL CREDENTIALS

If you are — or want to be — an agile practitioner, you may consider getting one or more agile certifications. The certification training alone can provide valuable information and the chance to practice agile processes — lessons you can use in your everyday work. Many organizations want to hire people with proven agile knowledge, so certification can also boost your career.

You can choose from a number of well-recognized, entry-level certifications, including the following:

- **Certified ScrumMaster (CSM):** The Scrum Alliance, a professional organization that promotes the understanding and use of scrum, offers a certification for scrum masters. The CSM requires a two-day training class, provided by a Certified Scrum Trainer (CST) and completing a CSM evaluation. CSM training provides an overall view of scrum and is a good starting point for people starting their agile journey. See <http://scrumalliance.org>.
- **Certified Scrum Product Owner (CSPO):** The Scrum Alliance also provides a certification for product owners. Like the CSM, the CSPO requires two days of training from a CST. CSPO training provides a deep dive into the product owner role. See <http://scrumalliance.org>.
- **Certified Scrum Developer (CSD):** For development team members, the Scrum Alliance offers the CSD. The CSD is a technical-track certification, requiring five days of training from a CST and passing an exam on agile engineering techniques. CSM or CSPO training can count toward a CSD; the remaining three days are a technical skills course. See <http://scrumalliance.org>.
- **PMI Agile Certified Practitioner (PMI-ACP):** The Project Management Institute (PMI) is the largest professional organization for project managers in the world. In 2012, PMI introduced the PMI-ACP certification. The PMI-ACP requires training,

general project management experience, experience working on agile projects, and passing an exam on your knowledge of agile fundamentals. See <http://pmi.org>.

## An overview of extreme programming

One popular approach to product development, specific to software, is extreme programming (XP). Extreme programming takes the best practices of software development to an extreme level. Created in 1996 by Kent Beck, with the help of Ward Cunningham and Ron Jeffries, the principles of XP were originally described in Beck's 1999 book, *Extreme Programming Explained* (Addison-Wesley Professional), which has since been updated.

The focus of extreme programming is customer satisfaction. XP teams achieve high customer satisfaction by developing features when the customer needs them. New requests are part of the development team's daily routine, and the team is empowered to deal with these requests whenever they crop up. The team organizes itself around any problem that arises and solves it as efficiently as possible.



TECHNICAL STUFF As XP has grown as a practice, XP roles have blurred. A typical project now consists of people in customer, management, technical, and project support groups. Each person may play a different role at different times.

## Discovering extreme programming principles

Basic approaches in extreme programming are based on Agile principles. These approaches are as follows:

- » **Coding is the core activity.** Software code not only delivers the solution but can also be used to explore problems. For example, a programmer can explain a problem using code.
- » **XP teams do lots of testing.** If doing just a little testing helps you identify some defects, a lot of testing will help you find more. In fact, developers don't start coding until they've worked out the success criteria for the requirement and designed the unit tests. A defect is not a failure of code; it's a failure to define the right test.

- » **Communication between customer and programmer is direct.** The programmer must understand the business requirement to design a technical solution.
- » **For complex systems, some level of overall design, beyond any specific function, is necessary.** In XP projects, the overall design is considered during regular *refactoring* — namely, using the process of systematically improving the code to enhance readability, reduce complexity, improve maintainability, and ensure extensibility across the entire code base.



REMEMBER You may find extreme programming combined with lean or scrum because the process elements are so similar that they marry well.

### ***Getting to know some extreme programming practices***

In XP, some practices are similar to other agile approaches, but others aren't. [Table 4-1](#) lists a few key XP practices, most of which are commonsense practices and many of which are reflected in agile principles.

### **TABLE 4-1 Key Practices of Extreme Programming**

<b><i>XP Practice</i></b>	<b><i>Underpinning Assumption</i></b>
Planning game	All members of the team should participate in planning. No disconnect exists between business and technical people.
Whole team	The customer needs to be collocated (physically located together) with the development team and be available. This accessibility enables the team to ask more minor questions, quickly get answers, and ultimately deliver a product more aligned with customer expectations.
Coding standards	Use coding standards to empower developers to make decisions and to maintain consistency throughout the product; don't constantly reinvent the basics of how to develop products in your organization. Standard code identifiers and naming conventions are two examples of coding standards.
System metaphor	When describing how the system works, use an implied comparison, a simple story that is easily understood (for instance, "the system is like cooking a meal"). This provides additional context that the team can fall back on in all product discovery activities and discussions.
Collective code ownership	The entire team is responsible for the quality of code. Shared ownership and accountability bring about the best designs and highest quality. Any engineer can modify another engineer's code to enable progress to continue.
Sustainable pace	Overworked people are not effective. Too much work leads to mistakes, which leads to more work, which leads to more mistakes. Avoid working more than 40 hours per week for an extended period of time.

Pair programming	Two people work together on a programming task. One person is strategic (the driver), and one person is tactical (the navigator). They explain their approach to each other. No piece of code is understood by only one person. Defects can more easily be found and fixed before merging and integrating code with the system.
Design improvement	Continuously improve design by refactoring code — removing duplications and inefficiencies within the code. A lean code base is simpler to maintain and operates more efficiently.
Simple design	The simpler the design, the lower the cost to change the software code.
Test-driven development (TDD)	Write automated customer acceptance and unit tests before you code anything. Write a test, run it, and watch it fail. Then write just enough code to make the test pass, refactoring until it does (red-green-clean). Test your success before you claim progress.
Continuous integration	Team members should be working from the latest code. Integrate code components across the development team as often as possible to identify issues and take corrective action before problems build on each other.
Small releases	Release value to the customer as often as possible. Some organizations release daily. Avoid building up large stores of unreleased code requiring extensive risky regression and integration efforts. Get feedback from your customer as early as possible, as often as possible.



**REMEMBER** Extreme programming intentionally pushes the limits of development customs by dramatically increasing the intensity of best practice rituals, which has resulted in a strong track record of XP improving development efficiency and success.

## Putting It All Together

All three agile approaches — lean, scrum, and extreme programming (XP) — have common threads. The biggest thing these approaches have in common is adherence to the Agile Manifesto and the 12 Agile Principles. [Table 4-2](#) shows a few more of the similarities among the three approaches.

**TABLE 4-2 Similarities between Lean, Scrum, and Extreme Programming**

<i>Lean</i>	<i>Scrum</i>	<i>Extreme Programming</i>
Engaging everyone	Cross-functional development team	Entire team Collective ownership
Optimizing the whole	Product increment	Test-driven development Continuous integration

In addition to more extensive agile frameworks and practices, scrum also accommodates a variety of accouterments that consistently increase success with agile projects. Just like a physical home is framed to support the plumbing, electrical, ventilation, and internal convenience features, scrum provides the framework for many other agile tools and techniques to do the job well. Here is a sampling, most of which you learn more about in the following chapters:

- » Product vision statement (elevator pitch, clear statement of direction for reaching the outer boundary of the project)
- » Product roadmap (a representation of the features required to achieve the product vision)
- » Velocity (a tool for scrum teams to plan the workload for each sprint and empirically predict the delivery of functionality long-term)
- » Release planning (establishing a specific mid-range goal, the trigger for releasing functionality to the market)
- » User stories (structuring requirements from an end-user's point of view to clarify business value)
- » Relative estimation (using self-correcting relative complexity and effort rather than inaccurate absolute measures, which give a false sense of precision)
- » Swarming (cross-functional teams working together on one requirement at a time until completion to get the job done faster)

# Chapter 5

## Agile Environments in Action

---

### IN THIS CHAPTER

- » **Creating your agile workspace**
- » **Rediscovering low-tech communication and using the right high-tech communication**
- » **Finding and using the tools you need**

Conjure up a mental picture of your current working environment. Perhaps it looks like the following setup. The IT team sits in cube city in one departmental area with the project manager somewhere within walking distance. You work with an offshore development team eight time zones away. The business customer is on the other side of the building. Your manager has a small office tucked away somewhere. Conference rooms are usually fully booked, and even if you were to get into one, someone would chase you out within the hour.

Your project documents are stored in folders on a shared drive. The development team gets at least 100 emails a day. The project manager holds a team meeting every week and, referring to the project plan, tells the developers what to work on. The project manager also creates a weekly status report and posts it on the shared drive. The product manager is usually too busy to talk to the project manager to review progress but periodically sends emails with some new thoughts about the application.

Although the description in the preceding paragraphs may not describe your particular situation, you can see something like it in any given corporate setting. Agile teams, however, execute projects in short, focused iterative cycles, relying on timely feedback from project team members. To operate and become more agile, your working environment is going to have to change.

This chapter shows you how to create a working space that facilitates

communication, one that will help you best become agile.

## ***Creating the Physical Environment***

Agile project teams flourish when scrum team members work closely together in an environment that supports the process. As noted in other chapters, the development team members are central to the success of agile projects. Creating the right environment for them to operate in goes a long way toward supporting their success. You can even hire people who specialize in designing optimal agile work environments.

### ***Collocating the team***

If at all possible, the scrum team needs to be *collocated* — that is, physically located together. When a scrum team is collocated, the following practices are possible and significantly increase efficiency and effectiveness:

- » Communicating face to face
- » Physically standing up — rather than sitting — as a group for the daily scrum meeting (this keeps meetings brief and on topic)
- » Using simple, low-tech tools for communication
- » Getting real-time clarifications from scrum team members
- » Being aware of what others are working on
- » Asking for help with a task
- » Supporting others with their tasks

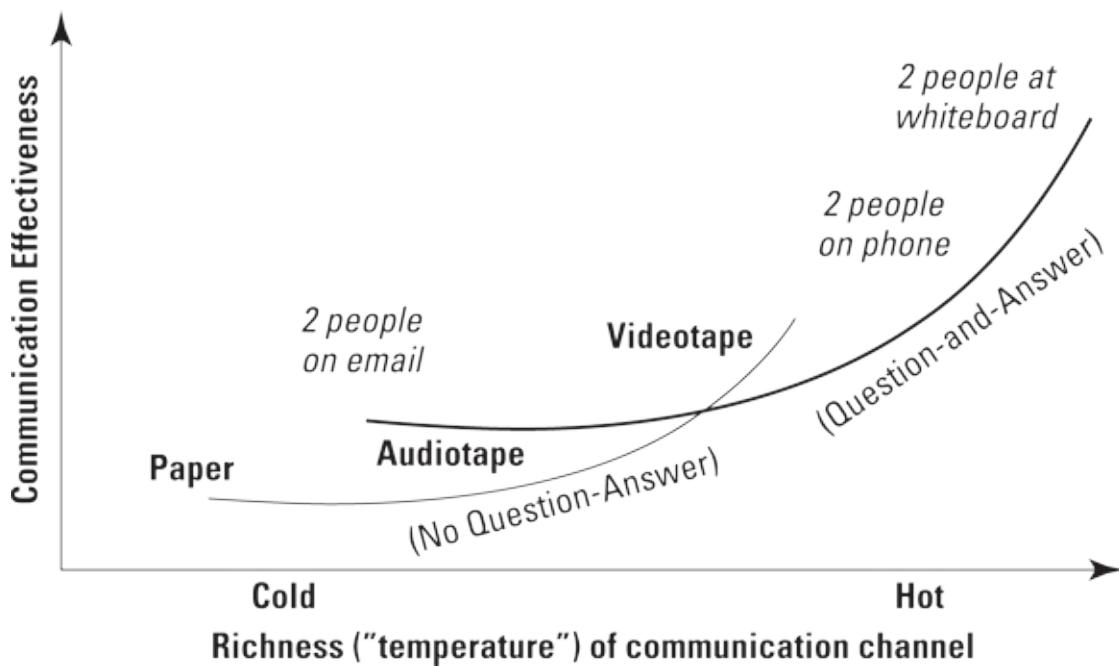
All these practices uphold agile processes. When everyone resides in the same area, it's much easier for one person to lean over, ask a question, and get an immediate answer. If the question is complex, a face-to-face conversation, with all the synergy it creates, is much more productive than an email exchange.



TECHNICAL STUFF This improved communication effectiveness is due to *communication fidelity* — the degree of accuracy between the meaning intended and the

meaning interpreted. Albert Mehrabian, Ph.D., a professor at UCLA, has shown that for complex, incongruent communication, 55 percent of meaning is conveyed by physical body language, 38 percent is conveyed through cultural-specific voice tonality interpretation, and only 7 percent is conveyed by words. That's something to keep in mind during your next voice-over IP or smartphone conference call to discuss the design nuances of a system that doesn't exist.

Alistair Cockburn, one of the Agile Manifesto signatories, created the graph in [Figure 5-1](#). This graph shows the effectiveness of different forms of communication. Notice the difference in effectiveness between paper communication and two people at a whiteboard — with collocation, you get the benefit of better communication.



[FIGURE 5-1:](#) Better communication through collocation.

## ***Setting up a dedicated area***

If the scrum team members are in the same physical place, you want to create as ideal a working environment for them as you can. The first step is to create a dedicated area.

Set up an environment where the scrum team can work in close physical proximity. If possible, the scrum team should have its own room, sometimes called a *project room* or a *scrum room*. The scrum team members create the

setup they need in this project room, putting whiteboards and bulletin boards on the walls and moving the furniture. By arranging the space for productivity, it becomes part of how they work. If a separate room isn't possible, a *pod* — with workspaces around the edges and a table or collaboration center in the middle — works well.

If you're stuck in cube city and can't tear down walls, ask for some empty cubes in a group and remove the dividing panels. Create a space that you can treat as your project room.



**REMEMBER** The right space allows the scrum team to be fully immersed in solving problems and crafting solutions.

The situation you have may be far from perfect, but it's worth the effort to see how close you can get to the ideal. Before you start an agile transition in your organization, ask management for the resources necessary to create an optimal condition. Resources will vary from project to project, but at a minimum, they can include whiteboards, bulletin boards, markers, pushpins, and sticky notes. You'll be surprised at how quickly the efficiency gains pay for the investment and more.

For example, with one client company, dedicating a project room and making a \$6,000 investment in multiple monitors for developers increased productivity, which saved the company almost two months and \$60,000 over the life of the project. That's a pretty good return on a simple investment. We show you how to quantify these savings early on in the project in [Chapter 13](#).

## ***Removing distractions***

The development team needs to focus, focus, focus. Agile methods are designed to create structure for highly productive work carried out in a specific way. The biggest threat to this productivity is distraction, such as ... hold on a minute, I need to take a call.

Okay, I'm back. The good news is that an agile team has someone dedicated to deflecting or eliminating distractions: the scrum master. Whether you're going to be taking on a scrum master role or some other role, you need to understand what sorts of distractions can throw the development team off course and how to handle them. [Table 5-1](#) is a list of common distractions

and do's and don'ts for dealing with distractions.

## TABLE 5-1 Common Distractions

<b>Distraction</b>	<b>Do</b>	<b>Don't</b>
Multiple projects	Do make sure that the development team is dedicated 100 percent to a single project at a time.	Don't fragment the development team between multiple projects, operations support, and special duties.
Multitasking	Do keep the development team focused on a single task, ideally developing one piece of functionality at a time. A task board can help keep track of the tasks in progress and quickly identify whether someone is working on multiple tasks at once.	Don't let the development team switch between requirements. Switching tasks creates a huge overhead (a minimum of 30 percent) in lost productivity.
Over-supervising	Do leave development team members alone after you collaborate on iteration goals; they can organize themselves. Watch their productivity skyrocket.	Don't interfere with the development team or allow others to do so. The daily scrum meeting provides ample opportunity to assess progress.
Outside influences	Do redirect any distractors. If a new task outside the sprint goal surfaces, ask the product owner to decide whether the task's priority is worth sacrificing sprint functionality.	Don't mess with the development team members and their work. They're pursuing the sprint goal, which is the top priority during an active sprint. Even a seemingly quick task can throw off work for an entire day.
Management	Do shield the development team from direct requests from management (unless management wants to give team members a bonus for their excellent performance).	Don't allow management to negatively affect the productivity of the development team. Make interrupting the development team the path of greatest resistance.



**REMEMBER** Distractions sap the development team's focus, energy, and performance. The scrum master needs strength and courage to manage and deflect interruptions. Every distraction averted is a step toward success.

## Going mobile

Judging by the “[Going mobile](#)” heading, you might have thought this section was about smartphone teleconferencing, but it isn’t. Agile project teams take a responsive approach, and scrum team members require an environment that helps them respond to the project needs of the day. An agile team environment should be mobile — literally:

- » Use movable desks and chairs so that people can move about and

reconfigure the space.

- » Get wirelessly connected laptops so that scrum team members can pick them up and move them about easily.
- » Have a large mobile whiteboard. Also see the next section on low-tech communication.

With this movable environment, scrum team members can configure and reconfigure their arrangement as needed. Given that scrum team members will be working with different members from day to day, mobility is important. Fixed furniture tends to dictate the communications that take place. Being mobile allows for freer collaboration and more freedom overall.

## ***Low-Tech Communicating***

When a scrum team is collocated, the members can communicate in person with ease and fluidity. Particularly when you begin your agile transition, you want to keep the communication tools low-tech. Rely on face-to-face conversations and good old-fashioned pen and paper. Low-tech promotes informality, allowing scrum team members to feel that they can change work processes and be innovative as they learn about the product.

The primary tool for communication should be face-to-face conversation. Tackling problems in person is the best way to accelerate production:

- » **Have short daily scrum meetings in person.** Some scrum teams stand throughout a meeting to discourage it from running longer than 15 minutes.
- » **Ask the product owner questions.** Also, make sure he or she is involved in discussions about product features to provide clarity when necessary. The conversation shouldn't end when planning ends.
- » **Communicate with your co-workers.** If you have questions about features, the project's progress, or integrating, communicate with co-workers. The entire development team is responsible for creating the product, and team members need to talk throughout the day.

As long as the scrum team is in close proximity, you can use physical and visual approaches to keep everyone on the same page. The tools should

enable everyone to see

- » The goal of the sprint
- » The functionality necessary to achieve the sprint goal
- » What has been accomplished in the sprint
- » What's coming next in the sprint
- » Who is working on which task
- » What remains to be done

Only a few tools are needed to support this low-tech communication:

- » A whiteboard or two (ideally, mobile — on wheels or lightweight). Nothing beats a whiteboard for collaboration. The scrum team can use one for brainstorming solutions or sharing ideas.
- » A huge supply of sticky notes in different colors (including poster-sized ones for communicating critical information you want readily visible — such as architecture, coding standards, and the project's definition of done).



TIP A personal favorite is giving each developer at least one tabletop dry erase/sticky note easel pad combination, with a lightweight easel. These low-cost tools facilitate communication fantastically.

- » Lots of colorful pens.
- » A sprint-specific task or kanban board (described in [Chapters 4](#) and [9](#)) for tracking progress tactility.

If you decide to have a sprint-specific kanban board, use sticky notes to represent *units of work* (features broken down into tasks). For your work plan, you can place sticky notes on a large surface (a wall or your second whiteboard), or you can use a kanban board with cards. You can customize a kanban board in many ways, such as using different-colored sticky notes for different types of tasks, red flag stickers for features that have an impediment, and team member stickers to easily see who is working on which task.



TECHNICAL STUFF

An *information radiator* is a tool that physically displays information to the scrum team and anyone else in the scrum team's work area. Information radiators include kanban boards, whiteboards, bulletin boards, *burndown charts*, which show the iteration's status, and any other sign with details about the project, the product, or the scrum team.

Basically, you move sticky notes or cards around the board to show the status (see [Figure 5-2](#)). Everyone knows how to read the board and how to act on what it shows. In [Chapter 9](#), you find out the details of what to put on the boards.

RELEASE GOAL:

SPRINT GOAL:

RELEASE DATE:

SPRINT REVIEW:

= User Story  
 = Task

TO DO	IN PROGRESS	ACCEPT	DONE

**FIGURE 5-2:** A scrum task board on a wall or whiteboard.



**TIP** Whatever tools you use, avoid spending time making things look perfectly neat and pretty. Formality in layout and presentation (what you might call *pageantry*) can give an impression that the work is tidy and elegant. However, the work is what matters, so focus your energy on activities that support the work.

## ***High-Tech Communicating***

Although collocation almost universally improves effectiveness, many scrum teams can't be collocated. Some projects have teammates scattered across multiple offices; others have offshore development teams around the world. If you have multiple, geographically scattered scrum teams, try first to reallocate existing talent to form scrum teams collocated within each geographic location. If this move isn't possible, don't give up on an agile transition. Instead, simulate collocation as much as possible.

When scrum team members work in different places, you have to make a greater effort to set up an environment that creates a sense of connectedness. To span distance and time zones, you need more sophisticated communication mechanisms.

### **DON'T REINVENT THE WHEEL!**

In the past, manufacturing processes often involved partially completed items being shipped to another location for completion. In these situations, the kanban board on a factory wall in the first location needed to be seen by shop floor management at the second location. Electronic kanban board software was developed to resolve this problem, but interestingly, the software looked like a literal kanban board on the wall and was used in the same way. Don't fix what's already working.

When determining which types of high-tech communication tools to support, first consider the loss of face-to-face discussions. Some tools you can use follow:

- » **Videoconferencing and webcams:** These tools can create a sense of being together. If you have to communicate remotely, at the very least

make sure you can see and hear each other clearly. Body language provides the majority of the message.

- » **Instant messaging:** Although instant messaging doesn't convey nonverbal communication, it is real time, accessible, and easy to use. Several people can also share a session and share files.
- » **Web-based desktop sharing:** Especially for the development team, sharing your desktop allows you to highlight issues and updates visually in real time. Seeing the problem is always better than just talking it out over the phone.
- » **Collaboration websites:** These sites allow you to do everything from sharing simple documentation so that everyone has the latest information to using a virtual whiteboard for brainstorming.



**TIP** Using a collaboration site (such as SharePoint, Confluence, and Google Drive) allows you to post documents that show the status of the sprint. When managers request status updates, you can simply direct them to the collaboration site to pull the information they need, on demand. By updating these documents daily, you provide managers with better information than they would have with formalized status reporting procedures under a traditional project management cycle. Avoid creating separate status reports for management; these reports duplicate information in the sprint burndowns and don't support production.



**WARNING** When you have a collaboration site with shared documentation, don't assume that everyone automatically understands everything in the documentation. Use a collaboration site to make sure everything is published, accessible, and transparent, but don't let it give your team a false sense of shared understanding.

## *Choosing Tools*

As noted throughout the chapter, low-tech tools are best suited for agile

projects, especially initially, while the scrum team becomes accustomed to the process. This section discusses a few points to consider when choosing agile tools: the purpose of the tool and organizational and compatibility constraints.

## ***The purpose of the tool***

When choosing tools, the primary question you need to ask is, “What is the purpose of the tool?” Tools should solve a specific problem and support agile processes, the focus of which is pushing forward with the work.

Above all, don’t choose anything more complicated than you need. Some tools are sophisticated and take time to learn before you can use them to be productive. If you’re working with a collocated scrum team, the training and adoption of agile practices can be enough of a challenge without adding a suite of complicated tools to the mix. If you’re working with a dislocated scrum team, introducing new tools can be even more difficult.



**WARNING** You can find a lot of agile-centric websites, software, and other tools on the market. Many are useful, but you shouldn’t invest in expensive agile tools in your early days of implementing agile. This investment is unnecessary and adds another level of complexity to adoption. As you go through the first few iterations and modify your approach, the scrum team will start identifying procedures that can be improved or need to change. One of these improvements might be the need for additional tools or replacement tools. When a need emerges naturally, from the scrum team, finding organizational support for purchasing the necessary tools is often easier because the need can be tied to a project issue.

## ***Organizational and compatibility constraints***

Beyond the initial considerations noted in the preceding section, the tools you choose must operate in your organization. Unless you’re using solely non-electronic tools, you’ll likely have to take into account organizational policies with respect to hardware, software, and services as well as cloud computing, security, and telephony systems.

If you’re part of a distributed organization, some scrum teams may not be able to support complex solutions, maintain the latest versions of desktop

software, or have the robust Internet bandwidth you take for granted.

The key to creating an agile environment for agile teams is to do so at the strategic organizational level. Agile teams drive agile projects, so enlist your organization's leadership early to provide tools that will empower your teams to succeed.

# Chapter 6

## Agile Behaviors in Action

---

### IN THIS CHAPTER

- » Setting up agile roles
- » Creating agile values in your organization
- » Transforming your team's philosophy
- » Sharpening important skills

In this chapter, you look at the behavioral dynamics that need to shift for your organization to benefit from the performance advantages that agile techniques enable. You find out about the different roles on an agile project and see how you can change a project team's values and philosophy about project management. Finally, we discuss some ways for a project team to hone key skills for agile project success.

### *Establishing Agile Roles*

In [Chapter 4](#), we describe scrum, one of the most popular agile frameworks in use today. The scrum framework defines common agile roles in an especially succinct manner. We use scrum terms to describe agile roles throughout this book. These roles are

- » Product owner
- » Development team member
- » Scrum master

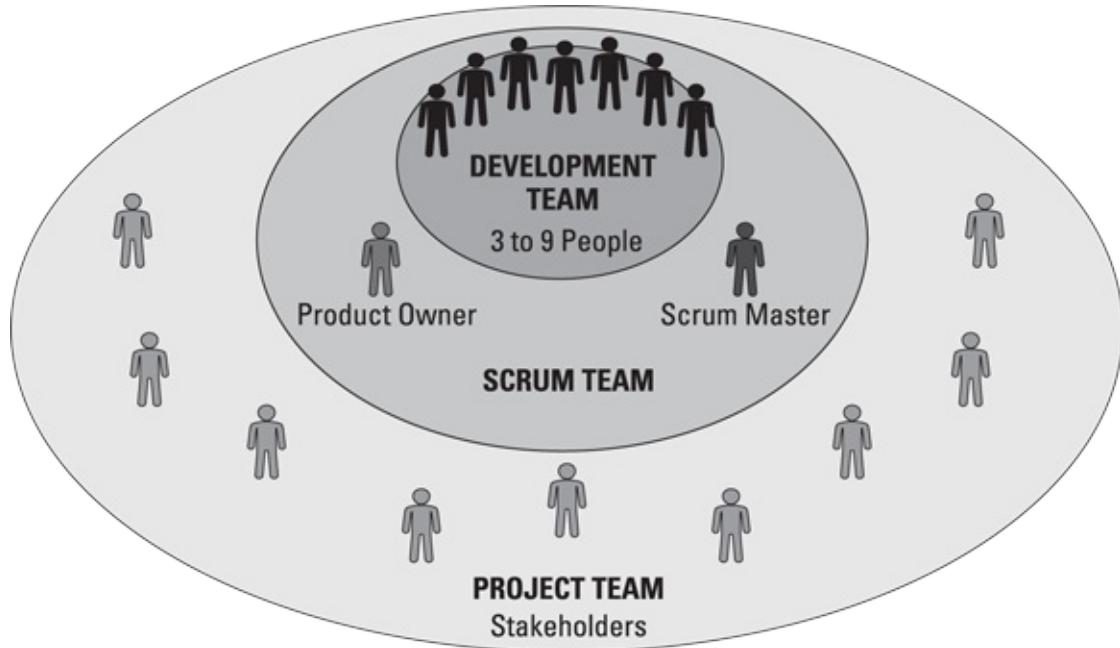
The product owner, development team, and scrum master together make up the *scrum team*. Each role is a peer to the others — no one is the boss of anyone else on the team.

The following roles are not part of the scrum framework but are still critically

important to agile projects:

- » Stakeholders
- » Agile mentor

The scrum team together with the stakeholders make up the *agile project team*. At the center of it all is the development team. The product owner and scrum master fulfill roles that ensure the development team's success. [Figure 6-1](#) shows how these roles and teams fit together. This section discusses these roles in detail.



[FIGURE 6-1:](#) Agile project team, scrum team, and development team.

## ***Product owner***

The product owner, sometimes called the *customer representative* in non-scrum environments, is responsible for bridging the gaps between the customer, business stakeholders, and the development team. The product owner is an expert on the product and the customer's needs and priorities. The product owner, who is a peer member of the scrum team, shields the development team from business distractions, works with the development team daily to help clarify requirements, and accepts completed work throughout the sprint in preparation for the sprint review.

Product owners make the decisions about what the product does and does not

include. Add to that the responsibility of deciding what to release to the market and when to do it, and you see that you need a smart and savvy person to fill this role.

On an agile project, the product owner will

- » Develop strategy and direction for the project and set long-and short-term goals.
- » Provide or have access to product expertise.
- » Understand and convey the customer's and other business stakeholders' needs to the development team.
- » Gather, prioritize, and manage product requirements.
- » Take responsibility for the product's budget and profitability.
- » Decide when to release completed functionality.
- » Work with the development team on a daily basis to answer questions and make decisions.
- » Accept or reject completed work — as it's completed — during the sprint.
- » Present the scrum team's accomplishments at the end of each sprint, before the development team demonstrates these accomplishments.

What makes a good product owner? Decisiveness. Good product owners understand the customer thoroughly and are empowered by the organization to make difficult business decisions every day. Although able to gather requirements from stakeholders, product owners are knowledgeable about the product in their own right. They can prioritize with confidence.

Good product owners interact well with the business stakeholder community, the development team, and the scrum master. They are pragmatic and able to make trade-offs based on reality. They are accessible to the development team and also ask for what they need. They are patient, especially with questions from the development team.

[Table 6-1](#) outlines the responsibilities and matching characteristics of a product owner.

## **TABLE 6-1 Characteristics of a Good Product Owner**

---

### **Responsibility A Good Product Owner ...**

Supplies project strategy and direction	Envisions the completed product Firmly understands company strategy
Provides product expertise	Has worked with similar products in the past Understands the needs of the people who will use the product
Understands customer and other stakeholder needs	Understands relevant business processes Creates a solid customer input and feedback channel Works well with business stakeholders
Manages and prioritizes product requirements	Is decisive Focuses on efficiency Remains flexible Turns stakeholder feedback into valuable, customer-focused functionality Is practical about prioritizing financially valuable features, high-risk features, and strategic system improvements Shields the development team from business distractions (competing stakeholder requests)
Is responsible for budget and profitability	Understands which product features can deliver the best return on investment Manages budgets effectively
Decides on release dates	Understands business needs regarding timelines
Works with development team	Is accessible for daily clarification of requirements Works with the development team to understand capabilities Works well with developers Adeptly describes product features
Accepts or rejects work	Understands requirements and ensures that completed features work correctly
Presents completed work at the end of each sprint	Clearly introduces the accomplishments of the sprint before the development team demonstrates the sprint's working functionality

The product owner takes on a great deal of business-related responsibility during the project. Although the project sponsor funds and owns the budget, the product owner manages how the budget is spent.

With a dedicated and decisive product owner, the development team has all the business support it needs to turn requirements into working functionality. The following section explains how the product owner helps ensure that the development team understands the product it will create.

## ***Development team member***

Development team members are the people who create the product. In

software development, programmers, testers, designers, writers, data engineers, and anyone else with a hands-on role in product development are development team members. With other types of product, the development team members may have different skills.

On an agile project, the development team is

- » Directly accountable for creating project deliverables.
- » Self-organizing and self-managing. The development team members determine their own tasks and how they want to complete those tasks.
- » Cross-functional. Collectively, the development team possesses all skills required to elaborate, design, develop, test, integrate and document requirements into working functionality.
- » Multi-skilled. Development team members are versatile — they're not tied to a single skill set. They have existing skills to immediately contribute at the beginning of the project, but they are also willing to learn new skills and to teach what they know to other development team members.
- » Ideally dedicated to one project for the duration of the project.
- » Ideally collocated. The team should be working together in the same area of the same office.

What makes a good development team member? Take a look at the team responsibilities and matching characteristics in [Table 6-2](#).

**TABLE 6-2 Characteristics of a Good Development Team Member**

<b>Responsibility</b>	<b>A Good Development Team Member ...</b>
Creates the product	Enjoys creating products Is skilled in more than one of the jobs necessary to create the product
Is self-organizing and self-managing	Exudes initiative and independence Understands how to work through impediments to achieve goals Coordinates the work to be done with the rest of the team
Is cross-functional	Has curiosity Willingly contributes to areas outside his or her mastery Enjoys learning new skills Enthusiastically shares knowledge
Is dedicated and collocated	Is part of an organization that understands the gains in efficiency and effectiveness associated with focused, collocated teams

---

---

The two other members of the scrum team, the product owner and the scrum master, help support the development team's efforts in creating the product. Whereas the product owner ensures that the development team is effective (working on the right things), the scrum master helps clear the way for the development team to work as efficiently as possible.

## **Scrum master**

A scrum master, sometimes called a *project facilitator* in non-scrum agile environments, is responsible for supporting the development team, clearing organizational roadblocks, and keeping processes true to agile principles.

A scrum master is different from a project manager. Teams using traditional project approaches work for a project manager. A scrum master, on the other hand, is a servant-leader peer who supports the team so that it is fully functional and productive. The scrum master role is an enabling role, rather than an accountability role. You can find more about servant leadership in [Chapter 14](#).

On an agile project, the scrum master will

- » Act as a process coach, helping the project team and the organization follow scrum values and practices.
- » Help remove project impediments — both reactively and proactively — and shield the development team from external interferences.
- » Foster close cooperation between stakeholders and the scrum team.
- » Facilitate consensus building within the scrum team.
- » Protect the scrum team from organizational distractions.



TIP We compare the scrum master to the aeronautical engineer whose job is to reduce drag on the aircraft. Drag is always there but can be reduced through innovative and proactive engineering. Likewise, all projects have organizational impediments creating drag on the team's efficiency, and there is always another constraint that can be identified and removed. One of the most significant parts of a scrum master's role is

removing roadblocks and preventing distractions to the development team's work. A scrum master who is good at these tasks is priceless to the project and to the team. If a development team has seven people, the effect of a good scrum master is times seven.

The product owner may never have participated in an agile project, but the scrum master likely has. As such, a scrum master may coach new product owners and development teams and does everything possible to help them succeed.

What makes a good scrum master? A scrum master doesn't need project manager experience. A scrum master is an expert in agile processes and can coach others. The scrum master must also work collaboratively with the product owner and the stakeholder community.



**TIP** Facilitation skills cut through the noise of group gatherings and ensure that everyone on the scrum team is focused on the right priority at the right time.

Scrum masters have strong communication skills, with enough organizational clout to secure the conditions for success by negotiating for the right environment, protecting the team from distractions, and removing impediments. Scrum masters are great facilitators and great listeners. They can negotiate their way through conflicting opinions and help the team help itself. Review the scrum master's responsibilities and matching characteristics in [Table 6-3](#).

### **TABLE 6-3 Characteristics of a Good Scrum Master**

<b><i>Responsibility</i></b>	<b><i>A Good Scrum Master ...</i></b>
Upholds scrum values and practices	Is an expert on scrum processes Is passionate about agile techniques
Removes roadblocks and prevents disruptions	Has organizational clout and can resolve problems quickly Is articulate, diplomatic, and professional Is a good communicator and a good listener Is firm about the development team's need to focus only on the project and the current sprint
Fosters close cooperation between external stakeholders and the scrum team	Looks at the needs of the project as a whole Avoids cliques and helps break down group silos

Facilitates consensus building	Understands techniques to help groups reach agreements
Is a servant-leader	Does not need or want to be in charge or be the boss Ensures that all members of the development team have the information they need to do the job, use their tools, and track progress Truly desires to help the scrum team



**TIP** Clout is not the same thing as authority. Organizations need to empower their scrum masters so they can influence change in the project team and organization, but clout involves earned respect, often through success and experience. Some types of clout that empower scrum masters come about through expertise (usually a niche knowledge), longevity (“I’ve been at the company a long time and know its history first hand”), charisma (“people generally like me”), or associations (“I know important people”). Don’t underestimate the value of a scrum master with clout.

The members of the scrum team — the product owner, development team, and scrum master — work together on the project every day.

As we mention earlier in the chapter, the scrum team plus stakeholders make up the project team. Sometimes stakeholders have less active participation than scrum team members but still can have considerable effect and provide a great deal of value to a project.

## GAINING CONSENSUS: THE FIST OF FIVE

Part of working as a team means agreeing on decisions as a team. An important part of being a scrum master is helping the team build consensus. We’ve all worked with groups where it was difficult to arrive at consensus, from how long a task would take to where to go for lunch. A quick, casual way to find out whether a group agrees with an idea is to use the *fist of five*, which appears similar to rock-paper-scissors.

On the count of three, each person holds up a number of fingers, reflecting the degree of comfort with the idea in question:

- 5: I love the idea.
- 4: I think it’s a good idea.
- 3: I can support the idea.
- 2: I have reservations, so let’s discuss.

### 1: I am opposed to the idea.

If some people have three, four, or five fingers up, and some have only one or two, discuss the idea. Find out why the people who support the idea think it will work, and what reservations the people who oppose the idea have. You want to get all group members showing at least three fingers — they don't need to love the idea, but they need to support it. The scrum master's consensus-building skills are essential for this task.

You can also quickly get an idea of consensus on a decision by asking for a simple thumb up (support), thumb down (don't support), or thumb to the side (undecided). It's quicker than a fist of five, and is great for answering yes-or-no questions.

## **Stakeholders**

Stakeholders are anyone with an interest in the project. They are not ultimately responsible for executing the product, but they provide input and are affected by the project's outcome. The group of stakeholders is diverse and can include people from different departments or even different companies.

On an agile project, stakeholders

- » Include the customer
- » May include technical people, such as infrastructure architects or system administrators
- » May include the legal department, account managers, salespeople, marketing experts, and customer service representatives
- » May include product or subject matter experts besides the product owner

Stakeholders may help provide key insights about the product and its use. Stakeholders might work closely with the product owner during the sprint, and will give feedback about the product during the sprint review at the end of each sprint.

Stakeholders and the part they play vary among projects and organizations. Almost all agile projects have stakeholders outside the scrum team.

Some projects also have agile mentors, especially projects with project teams that are new to agile processes.

## **Agile mentor**

A mentor is a great idea for any area in which you want to develop new expertise. The *agile mentor*, sometimes called an *agile coach*, is someone who has experience implementing agile projects and can share that experience with a project team. The agile mentor can provide valuable feedback and advice to new project teams and to project teams that want to perform at a higher level.

On an agile project, the agile mentor

- » Serves in a mentoring role only and is not part of the scrum team
- » Is often a person from outside the organization, and can provide objective guidance, without personal or political considerations
- » Is an agile expert with significant experience in implementing agile techniques and running agile projects of different sizes

You may want to think of an agile mentor the way you think of a golf coach. Most people use a golf coach not because they don't know how to play the game of golf but because a golf coach objectively observes things that a player engaged in the game never notices. Golf, like implementing agile techniques, is an exercise where small nuances make a world of difference in performance.

## *Establishing New Values*

Lots of organizations post their core values on the wall. In this section, however, we are talking about values that represent a way of working together every day, supporting each other, and doing whatever it takes to achieve the scrum team's commitments.

In addition to the values from the Agile Manifesto, the five core values for scrum teams are

- » Commitment
- » Courage
- » Focus
- » Openness
- » Respect

The following sections provide details about each of these values.

## ***Commitment***

Commitment implies engagement and involvement. On agile projects, the scrum team pledges to achieve specific goals. Confident that the scrum team will deliver what it promises, the organization mobilizes around the pledge to meet each goal.

Agile processes, including the idea of self-organization, provide people with all the authority they need to meet commitments. However, commitment requires a conscious effort. Consider the following points:

- » Scrum teams must be realistic when making commitments, especially for short sprints. It is easier, both logically and psychologically, to bring new features into a sprint than it is to take unachievable features out of a sprint.
- » Scrum teams must fully commit to goals. This includes having consensus among the team that the goal is achievable. After the scrum team agrees on a goal, the team does whatever it takes to reach that goal.
- » The scrum team is pragmatic but ensures that every sprint has a tangible value. Achieving a sprint goal and completing every item in the goal's scope are different. For example, a sprint goal of proving that a product can perform a specific action is much better than a goal stating that exactly seven requirements will be complete during the sprint. Effective scrum teams focus on the goal and remain flexible in the specifics of how to reach that goal.
- » Scrum teams are willing to be accountable for results. The scrum team has the power to be in charge of the project. As a scrum team member, you can be responsible for how you organize your day, the day-to-day work, and the outcome.

Consistently meeting commitments is central to using agile approaches for long-term planning. In [Chapter 13](#), you read about how to use performance to accurately determine project schedules and budgets.

## ***Courage***

We all experience fear. We all have certain things we don't want to do, whether asking a team member to explain something we don't understand or confronting the boss. Embracing agile techniques is a change for many organizations. Successfully making changes requires courage in the face of resistance. Following are some tips that foster courage:

- » **Realize that the processes that worked in the past won't necessarily work now.** Sometimes you need to remind people of this fact. If you want to be successful with agile techniques, your everyday work processes need to change to improve.
- » **Be ready to buck the status quo.** The status quo will push back. Some people have vested interests and will not want to change how they work.
- » **Temper challenge with respect.** Senior members of the organization might be especially resistant to change; they often created the old rules for how things were done. Now you're challenging those rules. Respectfully remind these individuals that you can achieve the benefits of agile techniques only by following the 12 agile principles faithfully. Ask them to give change a try.
- » **Embrace the other values.** Have the courage to make commitments and stand behind those commitments. Have the courage to focus and tell distractors "no." Have the courage to be open and to acknowledge that there is always an opportunity to improve. And have the courage to be respectful and tolerant of other people's views, even when they challenge your views.

As you replace your organization's antiquated processes with more modern approaches, expect to be challenged. Take on that challenge; the rewards can be worth it in the end.

## ***Focus***

Working life is full of distractions. Plenty of people in your organization would love to use your time to make their day easier. Disruptions, however, are costly. Jonathan Spira, from the consulting firm Basex, published a report called "The Cost of Not Paying Attention: How Interruptions Impact Knowledge Worker Productivity." His report details how businesses in the United States lose close to \$600 billion a year through workplace distractions.

Scrum team members can help change those dysfunctions by insisting on an environment that allows them to focus. To reduce distractions and increase productivity, scrum team members can

- » **Physically separate themselves from company distractors.** One of our favorite techniques for ensuring high productivity is to find an annex away from the company's core offices and have that be the scrum team's work area. Sometimes the best defense is distance.
- » **Ensure that you're not spending time on activities unrelated to the sprint goal.** If someone tries to distract you from the sprint goal with something that "has to be done," explain your priorities. Ask, "How will this request move the sprint goal forward?" This simple question can push a lot of activities off the to-do list.
- » **Figure out what needs to be done and do only that.** The development team determines the tasks necessary to achieve the sprint goal. If you're a development team member, use this ownership to drive your focus to the priority tasks at hand.
- » **Balance focused time with accessibility to the rest of the scrum team.** Francesco Cirillo's Pomodoro technique — splitting work into 25-minute time blocks, with breaks in between — helps achieve balance between focus and accessibility. We often recommend giving development team members noise-canceling headsets, the wearing of which is a "do not disturb" sign. However, we also suggest a team agreement that all scrum team members have a minimum set of office hours in which they are available for collaboration.
- » **Check that you're maintaining your focus.** If you're unsure of whether you are maintaining focus — it can be hard to tell — go back to the basic question, "Are my actions consistent with achieving the overall goal and the near-term goal (such as completing the current task)?"

As you can see, task focus is not a small priority. Extend the effort upfront to create a distraction-free environment that helps your team succeed.

## ***Openness***

Secrets have no place on an agile team. If the team is responsible for the result of the project, it only makes sense that they have all the facts at their

disposal. Information is power, and ensuring that everyone has access to the information necessary to make the right decisions requires a willingness to be transparent. To leverage the power of openness, you can

- » **Ensure that everyone on the team has access to the same information.** Everything from the vision for the project down to the smallest detail about the status of tasks needs to be in the public domain as far as the team is concerned. Use a centralized repository as the single source for information, and then avoid the distraction of “status reporting” by putting all status (burndowns, impediment list, and so forth) and information in this one place. We often send a link to this repository to the project stakeholders and say, “All the information we have is a click away. There is no faster way to get updated.”
- » **Be open and encourage openness in others.** Team members must feel free to speak openly about problems and opportunities to improve, whether the issues are something that they’re dealing with themselves or see elsewhere in the team. Openness requires trust within the team, and trust takes time to develop.
- » **Defuse internal politics by discouraging gossip.** If someone starts talking to you about what another team member did or didn’t do, ask him or her to take the issue to the person who can resolve it. Don’t gossip yourself. Ever.
- » **Always be respectful.** Openness is never an excuse to be destructive or mean. Respect is critical to an open team environment.

Small problems unaddressed often grow to become crises. Use an open environment to benefit from the input of the entire team and ensure that your development efforts are focused on the project’s true priorities.

## ***Respect***

Each individual on the team has something important to contribute. Your background, education, and experiences have a distinctive influence on the team. Share your uniqueness and look for, and appreciate, the same in others. You encourage respect when you

- » **Foster openness.** Respect and openness go hand in hand. Openness without respect causes resentment; openness with respect generates trust.

- » **Encourage a positive work environment.** Happy people tend to treat one another better. Encourage positivity, and respect will follow.
- » **Seek out differences.** Don't just tolerate differences; try to find them. The best solutions come from diverse opinions that have been considered and appropriately challenged.
- » **Treat everyone on the team with the same degree of respect.** All team members should be accorded the same respect, regardless of their role, level of experience, or immediate contribution. Encourage everyone to give his or her best.



**REMEMBER** Respect is the safety net that allows innovation to thrive. When people feel comfortable raising a wider range of ideas, the final solution can improve in ways that would never be considered without a respectful team environment. Use respect to your team's advantage.

## *Changing Team Philosophy*

An agile development team operates differently from a team using a waterfall approach. Development team members must change their roles based on each day's priorities, organize themselves, and think about projects in a whole new way to achieve their commitments.

To be part of a successful agile project, development teams should embrace the following attributes:

- » **Dedicated team:** Each scrum team member works only on the project assigned to the scrum team, and not with outside teams or projects. Projects may finish and new projects may start, but the team stays the same.
- » **Cross-functionality:** The willingness and ability to work on different types of tasks to create the product.
- » **Self-organization:** The ability and responsibility to determine how to go about the work of product development.
- » **Self-management:** The ability and responsibility to keep work on track.

- » **Size-limited teams:** Right-size development teams to ensure effective communication. Smaller is better; the development team should never be larger than nine people.
- » **Ownership:** Take initiative for work and responsibility for results.

The following sections look at each of these ideas in more detail.

## Dedicated team

A traditional approach to resource allocation (we prefer the term *talent allocation*) is to allocate portions of team members' time across multiple teams and projects to get to full 100 percent utilization to justify the expense of employing team members. For management, knowing that all hours of the week are accounted for and justified is gratifying. However, the result is lower productivity due to continual *context switching* — the cost associated with cognitive demobilization and remobilization to switch from one task to another.

Other common talent allocation practices include moving a team member from team to team to temporarily fill a skill gap or a manpower gap, and tasking a team with multiple projects at once. These tactics are often employed to try to do more with less, but all the input variances make it nearly impossible to predict outputs.

These approaches have similar results: a significant decrease in productivity and an inability to extrapolate performance. Studies clearly show a minimum of 30 percent increase in the time required to complete projects run in parallel instead of serially.



TECHNICAL STUFF *Thrashing* is another term for context switching between tasks. Avoid thrashing by dedicating team members to a single project at a time.

The following results occur when you dedicate scrum teams to work on only one project at a time:

- » **More accurate release projections:** Because the same people are consistently doing the same tasks every sprint with the same amount of time allocated to the project from sprint to sprint, scrum teams can

accurately and empirically extrapolate how long it will take to complete their remaining backlog items with more certainty than traditional splintered approaches.

- » **Effective, short iterations:** Sprints are short because the shorter the feedback loop, the more quickly scrum teams can respond to feedback and changing needs. There just isn't enough time for thrashing team members between projects.
- » **Fewer and less costly defects:** Context switching results in more defects because distracted developers produce lower quality functionality. It costs less to fix something while it is still fresh in your mind (during the sprint) than later, when you have to try to remember the context of what you were working on. Studies show that defects cost 6.5 times more to fix after the sprint ends and you've moved on to other requirements, 24 times more to fix when preparing for release, and 100 times more to fix after the product is in production.



TIP If you want more predictability, higher productivity, and fewer defects, dedicate your scrum team members. We've found this to be one of the highest factors of agile transition success.

## Cross-functionality

On traditional projects, experienced team members are often typecast as having a single skill. For example, a .NET programmer may always do .NET work, and a tester may always do quality control work. Team members with complementary skills are often considered to be part of separate groups, such as the programming group or the testing group.

Agile approaches bring the people who create products together into a cohesive group — the development team. People on agile development teams try to avoid titles and limited roles. Development team members may start a project with one skill, but learn to perform many different jobs throughout the project to help create the product.

Cross-functionality makes development teams more efficient. For example, suppose a daily scrum meeting uncovers testing as the highest priority task to

complete the requirement. A programmer might help test to finish the task quickly. When the development team is cross-functional, it can *swarm* on product features, with as many people working on a single requirement as possible, to quickly complete the feature.

Cross-functionality also helps eliminate single points of failure. Consider traditional projects, where each person knows how to do one job. When a team member gets sick, goes on vacation, or leaves the company, no one else may be capable of doing his or her job. The tasks that person was doing are delayed. By contrast, cross-functional agile development team members are capable of doing many jobs. When one person is unavailable, another can step in.

Cross-functionality encourages each team member to

- » **Set aside the narrow label of what he or she can do.** Titles have no place on an agile team. Skills and an ability to contribute are what matter. Start thinking of yourself as a Special Forces commando — knowledgeable enough in different areas that you can take on any situation.
- » **Work to expand skills.** Don't work only in areas you already know. Try to learn something new each sprint. Techniques such as *pair programming* — where two developers work together to code one item — or shadowing other developers can help you learn new skills quickly and increase overall product quality.
- » **Step up to help someone who has run into a roadblock.** Helping someone with a real-world problem is a great way to learn a new skill.
- » **Be flexible.** A willingness to be flexible helps to balance workloads and makes the team more likely to reach its sprint goal.

With cross-functionality in place, you avoid waiting for key people to work on tasks. Instead, a motivated, even if somewhat less knowledgeable, development team member can work on a piece of functionality today. That development team member learns and improves, and the workflow continues to be balanced.

One big payoff of cross-functionality is that the development team completes work quickly. Post-sprint review afternoons are often celebration

time. Go to the movies together. Head to the beach or the bowling alley. Go home early.

## **Self-organization**

Agile techniques emphasize self-organizing development teams to take advantage of development team members' varied knowledge and experience.



**REMEMBER** If you've read [Chapter 2](#), you may recall agile principle #11: The best architectures, requirements, and designs emerge from self-organizing teams.

Self-organization is an important part of being agile. Why? In a word: ownership. Self-organized teams are not complying with orders from others; they own the solution developed and that makes a huge difference in team member engagement and solution quality.

For development teams used to a traditional command-and-control project management model, self-organization may take some extra effort at first. Agile projects do not have a project manager to tell the development team what to do. Instead, self-organizing development teams

- » **Commit to their own sprint goals.** At the beginning of each sprint, the development team works with the product owner to identify an objective it can reach, based on project priorities.
- » **Identify their tasks.** Development team members determine the tasks necessary to meet each sprint goal. The development team works together to figure out who takes on which task, how to get the work done, and how to address risks and issues.
- » **Estimate the effort necessary for requirements and related tasks.** The development team knows the most about how much effort it will take to create specific product features.
- » **Focus on communication.** Successful agile development teams hone their communication skills by being transparent, communicating face-to-face, being aware of nonverbal communication, participating, and listening.



TIP The key to communication is clarity. With complex topics, avoid one-way, potentially ambiguous modes of communication, such as email. Face-to-face communication prevents misunderstandings and frustration. You can always summarize the conversation in a quick email later if details need to be retained.

- » **Collaborate.** Getting the input of a diverse scrum team almost always improves the product but requires solid collaboration skills. Collaboration is the foundation of an effective agile team.



REMEMBER No successful project is an island. Collaboration skills help scrum team members take risks with ideas and bring innovative solutions to project problems. A safe and comfortable environment is a cornerstone of a successful agile project.

- » **Decide with consensus.** For maximum productivity, the entire development team must be on the same page and committed to the goal at hand. The scrum master often plays an active role in building consensus, but the development team ultimately takes responsibility for reaching agreement on decisions, and everyone owns the decisions.
- » **Actively participate.** Self-organization may be challenging for the shy. All development team members must actively participate. No one is going to tell the development team what to do to create the product. The development team members tell themselves what to do. And when. And how.



TIP In our agile coaching experience, we've heard new agile development team members ask questions like, "So, what should I do now?" A good scrum master answers by asking the developer what he or she needs to do to achieve the sprint goal, or by asking the rest of the development team what they suggest. Answering questions with questions can be a

helpful way to guide a development team toward being self-organizing.

Being part of a self-organizing development team takes responsibility, but it also has its rewards. Self-organization gives development teams the freedom to succeed. Self-organization increases ownership, which can result in better products, which can help development team members find more satisfaction in their work.

## ***Self-management***

Self-management is closely related to self-organization. Agile development teams have a lot of control over how they work; that control comes with the responsibility for ensuring the project is successful. To succeed with self-management, development teams

- » **Allow leadership to ebb and flow.** On agile projects, each person on the development team has the opportunity to lead. For different tasks, different leaders will naturally emerge; leadership will shift throughout the team based on skill expertise and previous experiences.
- » **Rely on agile processes and tools to manage the work.** Agile methods are tailored to make self-management easy. With an agile approach, meetings have clear purposes and time limits, and artifacts expose information but rely on minimal effort to create and maintain. Taking advantage of these processes allows development teams to spend most of their time creating the product.
- » **Report progress regularly and transparently.** Each development team member is responsible for accurately updating work status on a daily basis. Luckily, progress reporting is a quick task on agile projects. In [Chapter 9](#), you find out about burndown charts, which provide status but only require a few minutes each day to update. Keeping status current and truthful makes planning and issue management easier.
- » **Manage issues within the development team.** Many obstacles can arise on a project: Development challenges and interpersonal problems are a couple of examples. The development team's first point of escalation for most issues is the development team itself.
- » **Create a team agreement.** Development teams sometimes make up a team agreement, a document that outlines the expectations each team member will commit to meet. Working agreements provide a shared

understanding of behavioral expectations and empower the facilitator to keep the team on track according to what they've already agreed together.

- » **Inspect and adapt.** Figure out what works for your team. Best practices differ from team to team. Some teams work best by coming in early; others work best by coming in late. The development team is responsible for reviewing its own performance and identifying techniques to continue and techniques to change.
- » **Actively participate.** As with self-organization, self-management works only when development team members join in and commit to guiding the project's direction.



**TIP** The development team is primarily responsible for self-organization and self-management. However, the scrum master can assist the development team in a number of ways. When development team members look for specific directions, the scrum master can remind them that they have the power to decide what to do and how to do it. If someone outside the development team tries to give orders, insist on tasks, or dictate how to create the product, the scrum master can intervene. The scrum master can be a powerful ally in the development team's self-organization and self-management.

## ***Size-limited teams***

Agile development teams are intentionally small. A small development team is a nimble team. As the development team size grows, so does the overhead associated with orchestrating task flow and communication flow.

Ideally, agile development teams have the least number of people necessary to be self-encapsulated (can do everything necessary to produce the product) and not have single points of failure. To have skill coverage, teams typically won't be any smaller than three people. Statistically, scrum teams are fastest with six developers, and cheapest with four to five developers. Keeping the development team size between three and nine people helps teams act as cohesive teams, and avoids creating subgroups, or *silos*.

Limiting development team size

- » Encourages diverse skills to be developed
- » Facilitates good team communication
- » Maintains the team in a single unit
- » Promotes joint code ownership, cross-functionality, and face-to-face communication

When you have a small development team, a similarly limited and focused project scope follows. Development team members are in close contact throughout the day as tasks, questions, and peer reviews flow back and forth among teammates. This cohesiveness ensures consistent engagement, increases communication, and reduces project risk.

When you have a large project and a correspondingly large development team, split the work between multiple scrum teams. For more on scaling agile projects across the enterprise, see [Chapter 17](#).

## **Ownership**

Being part of a cross-functional, self-organized, self-managing development team requires responsibility and ownership. The top-down management approaches on traditional projects do not always foster the maturity of ownership necessary for taking responsibility for projects and results. Even seasoned development team members may need to adjust their behavior to get used to making decisions on agile projects.

Development teams can adapt behavior and increase their level of ownership by doing the following:

- » **Take initiative.** Instead of waiting for someone else to tell you what to work on, take action. Do what is necessary to help meet commitments and goals.
- » **Succeed and fail as a team.** On agile projects, accomplishments and failures alike belong to the project team. If problems arise, be accountable as a group, rather than finding blame. When you succeed, recognize the group effort necessary for that success.
- » **Trust the ability to make good decisions.** Development teams can make mature, responsible, and sound decisions about product development. This takes a degree of trust as team members become accustomed to

having more control in a project.

Behavioral maturity and ownership doesn't mean that agile development teams are perfect. Rather, they take ownership for the scope they commit to, and they take responsibility for meeting those commitments. Mistakes happen. If they don't, you aren't pushing yourself outside your comfort zone. A mature development team identifies mistakes honestly, accepts responsibility for mistakes openly, and learns and improves from its mistakes consistently.

## Part 3

# Agile Planning and Execution

## **IN THIS PART ...**

Follow the Roadmap to Value, from vision to execution.

Define and estimate requirements.

Create working functionality and showcase it in iterations.

Inspect your work and adapt your processes for continuous improvement.

# Chapter 7

## Defining the Product Vision and Product Roadmap

---

### IN THIS CHAPTER

- » Planning agile projects
- » Establishing the product vision
- » Creating features and a product roadmap

To start, let's dispel a common myth. If you've heard that agile projects don't include planning, dismiss that thought right now. You will plan not only the overall project but also every release, every sprint, and every day. Planning is fundamental to agile project success.

If you're a project manager, you probably do the bulk of your planning at the beginning of a project. You may have heard the phrase, "Plan the work, then work the plan," which sums up non-agile project management approaches.

Agile projects, in contrast, involve planning upfront and throughout the entire project. By planning at the last responsible moment, right before an activity starts, you know the most about that activity. This type of planning, called *just-in-time planning* or a *situationaly informed strategy*, is a key to agile project success. Agile teams plan as much as, if not more than, traditional project teams. However, agile planning is more evenly distributed throughout the project and is done by the entire team that will be working on the project.



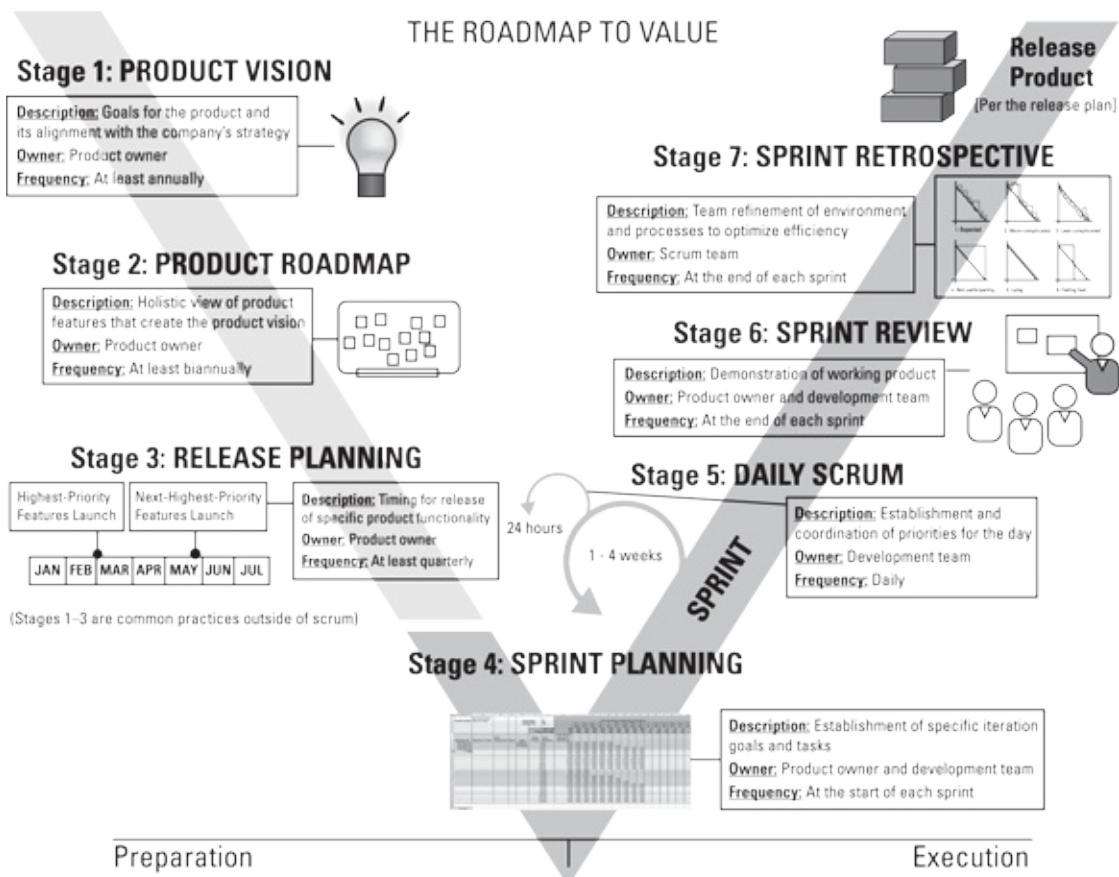
TECHNICAL STUFF Helmuth von Moltke, a nineteenth-century German field marshal and military strategist, once said, "No plan survives contact with the enemy." That is, in the heat of a battle — much like in the thick of a project — plans always change. The agile focus on just-in-time planning allows

you to accommodate real situations and to be well informed as you plan specific tasks.

This chapter describes how just-in-time planning works with agile projects. You also go through the first two steps of planning an agile project: creating the product vision and the product roadmap.

## Agile Planning

Planning happens at a number of points in an agile project. A great way to look at the planning activities in agile projects is with the Roadmap to Value. [Figure 7-1](#) shows the roadmap as a whole.



**FIGURE 7-1:** Stages of agile planning and execution with the Roadmap to Value.

The Roadmap to Value has seven stages:

- » In stage 1, the product owner identifies the *product vision*. The product vision is your project's destination or end goal. The product vision

includes the outer boundary of what your product will be, how the product is different from the competition, how the product will support your company or organization's strategy, who will use the product, and why people will use the product. On longer projects, revisit the product vision at least once a year.

- » In stage 2, the product owner creates a *product roadmap*. The product roadmap is a high-level view of the product requirements, with a general time frame for when you will develop those requirements. It also gives context to the vision by showing the tangible features that will be produced during the project. Identifying product requirements and then prioritizing and roughly estimating the effort for those requirements allow you to establish requirement themes and identify requirement gaps. The product owner, with support from the development team, should revise the product roadmap at least biannually.
- » In stage 3, the product owner creates a release plan. The *release plan* identifies a high-level timetable for the release of working functionality to the customer. The release serves as a mid-term boundary against which the scrum team can mobilize. An agile project will have many releases, with the highest-priority features appearing first. You create a release plan at the beginning of each release, which is usually at least quarterly. Find out more about release planning in [Chapter 8](#).
- » In stage 4, the product owner, the development team, and the scrum master will plan iterations, also called sprints, and start creating the product functionality in those sprints. *Sprint planning* sessions take place at the start of each sprint. During sprint planning, the scrum team determines a sprint goal, which establishes the immediate boundary of work that the team forecasts to accomplish during the sprint, with requirements that support the goal and can be completed in the sprint. The scrum team also outlines how to complete those requirements. Get more information about sprint planning in [Chapter 8](#).
- » In stage 5, the development team has *daily scrum* meetings during each sprint to coordinate the day's priorities. In the daily scrum meeting, you discuss what you completed yesterday, what you will work on today, and any roadblocks you have, so that you can address issues immediately. Read about daily scrums in [Chapter 9](#).

- » In stage 6, the scrum team holds a *sprint review* at the end of every sprint. In the sprint review, you demonstrate the working functionality to the product stakeholders. Find out how to conduct sprint reviews in [Chapter 10](#).
- » In stage 7, the scrum team holds a *sprint retrospective*. The sprint retrospective is a meeting where the scrum team discusses the completed sprint with regard to their processes and environment, and makes plans for process improvements in the next sprint. Like the sprint review for inspecting and adapting the product, a sprint retrospective is held at the end of every sprint to inspect and adapt your processes and environment. Find out how to conduct sprint retrospectives in [Chapter 10](#).

Each stage in the Roadmap to Value is repeatable, and each stage contains planning activities. Agile planning, like agile development, is iterative.

## **Progressive elaboration**

During each stage in an agile project, you plan only as much as you need to plan. In the early stages of your project, you plan widely and holistically to create a broad outline of how the product will shape up over time. In later stages, you narrow your planning and add more details to ensure success in the immediate development effort. This process is called a *progressive elaboration of requirements*.

Planning broadly at first and in detail later, when necessary, prevents you from wasting time on planning lower-priority product requirements that may never be implemented. This model also lets you add high-value requirements during the project without disrupting the development flow.

The more just-in-time your detailed planning is, the more efficient your planning process becomes.



**REMEMBER** Some studies show customers rarely or never use 64 percent of the features in an application. In the first few development cycles of an agile project, you complete features that have the highest priority and that people *will* use. Typically, you release those groups of features as early as possible to gain market share through first-mover advantage; receive

customer feedback for viability; monetize functionality early to optimize return on investment (ROI); and avoid internal and external obsolescence.

## ***Inspect and adapt***

Just-in-time planning brings into play two fundamental tenets of agile techniques: inspect and adapt. At each stage of a project, you need to look at the product and the process (inspect) and make changes as necessary (adapt).

Agile planning is a rhythmic cycle of inspecting and adapting. Consider the following:

- » Each day during the sprint, the product owner provides feedback to help improve the product as the development team creates the product.
- » At the end of each sprint, in the sprint review, stakeholders provide feedback to further improve the product.
- » Also at the end of each sprint, in the sprint retrospective, the scrum team discusses the lessons it learned during the past sprint to improve the development process.
- » After a release, the customers can provide feedback for improvement. Feedback might be direct, when a customer contacts the company about the product, or indirect, when potential customers either do or don't purchase the product.

Inspect and adapt, together, are fantastic tools for delivering the right product in the most efficient manner.



**REMEMBER** At the beginning of a project, you know the least about the product you're creating, so trying to plan fine details at that time just doesn't work. Being agile means you do the detailed planning when you need it, and immediately develop the specific requirements you defined with that planning.

Now that you know a little more about how agile planning works, it's time to complete the first step in an agile project: defining the product vision.

# Defining the Product Vision

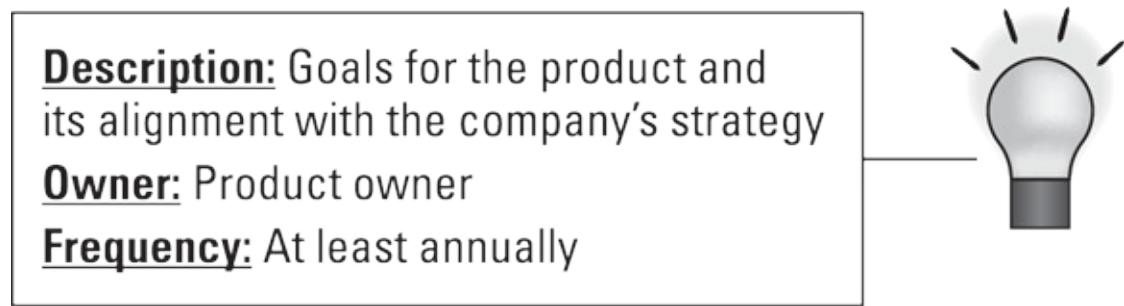
The first stage in an agile project is defining your product vision. The *product vision statement* is an elevator pitch, or a quick summary, to communicate how your product supports the company's or organization's strategies. The vision statement must articulate the end state for the product.

The product might be a commercial product for release to the marketplace or an internal solution that will support your organization's day-to-day functions. For example, say your company is XYZ Bank and your product is a mobile banking application. What company strategies does a mobile banking application support? How does the application support the company's strategies? Your vision statement clearly and concisely links the product to your business strategy.

[Figure 7-2](#) shows how the vision statement — stage 1 of the Roadmap to Value — fits with the rest of the stages and activities in an agile project.

## A common agile practice

### Stage 1: PRODUCT VISION



[FIGURE 7-2:](#) The product vision statement as part of the Roadmap to Value.

The product owner is responsible for knowing about the product, its goals, and its requirements throughout the project. For those reasons, the product owner creates the vision statement, although other people may have input. After the vision statement is complete, it becomes a guiding light, the “what we are trying to achieve” statement that the development team, scrum master, and stakeholders refer to throughout the project.

When creating a product vision statement, follow these four steps:

1. **Develop the product objective.**
2. **Create a draft vision statement.**
3. **Validate the vision statement with product and project stakeholders and revise it based on feedback.**
4. **Finalize the vision statement.**

The look of a vision statement follows no hard-and-fast rules. However, anyone involved with the project, from the development team to the CEO, should be able to understand the statement. The vision statement should be internally focused, clear, nontechnical, and as brief as possible. The vision statement should also be explicit and avoid marketing fluff.

## ***Step 1: Developing the product objective***

To write your vision statement, you must understand and be able to communicate the product's objective. You need to identify the following:

- » **Customer:** Who will use the product? This question might have more than one answer.
- » **Key product goals:** How will the product benefit the company that is creating it? The goals may include benefits for a specific department in your company, such as customer service or the marketing department, as well as the company as a whole. What specific company strategies does the product support?
- » **Need:** Why does the customer need the product? What features are critical to the customer?
- » **Competition:** How does the product compare with similar products?
- » **Primary differentiation:** What makes this product different from the status quo or the competition or both?

## ***Step 2: Creating a draft vision statement***

After you have a good grasp of the product's objective, create a first draft of your vision statement.

You can find many templates for a product vision statement. For an excellent guide to defining the overall product vision, see *Crossing the Chasm*, by Geoffrey Moore (published by HarperCollins), which focuses on how to

bridge the gap (chasm) between early adopters of new technologies and the majority who follow.

The adoption of any new product is a gamble. Will users like the product? Will the market take to the product? Will there be an adequate return on investment for developing the product? In *Crossing the Chasm*, Moore describes how early adopters are driven by vision, whereas the majority are skeptical of visionaries and interested in down-to-earth issues of quality, product maintenance, and longevity.



TECHNICAL STUFF *Return on investment*, or ROI, is the benefit or value a company gets from paying for something. ROI can be quantitative, such as the additional money ABC Products makes from selling widgets online after investing in a new website. ROI can also be something intangible, such as better customer satisfaction for XYZ Bank customers who use the bank's new mobile banking application.

By creating your vision statement, you help convey your product's quality, maintenance needs, and longevity.

Moore's product vision approach is pragmatic. In [Figure 7-3](#), we construct a template based on Moore's approach to more explicitly connect the product to the company's strategies. If you use this template for your product vision statement, it will stand the test of time as your product goes from early adoption to mainstream usage.

## Vision Statement for Product

For \_\_\_\_\_ (target customer)  
who \_\_\_\_\_ (needs)  
the \_\_\_\_\_ (product name)  
is a \_\_\_\_\_ (product category)  
that \_\_\_\_\_ (product benefit, reason to buy)  
Unlike \_\_\_\_\_ (competitors)  
our product \_\_\_\_\_ (differentiation/value proposition)

**FIGURE 7-3:** Expansion of Moore's template for a vision statement.



**TIP** One way to make your product vision statement more compelling is to write it in the present tense, as if the product already exists. Using present tense helps readers imagine the product in use.

Using our expansion of Moore's template, a vision statement for a mobile banking application might look like the following:

**For** XYZ Bank customers  
**who** want access to banking capability while on the go,  
**the** MyXYZ  
**is a** mobile application  
**that** allows secure, on-demand banking, 24 hours a day.  
**Unlike** online banking from your home or office computer,  
**our product** allows users immediate access,  
**which supports our strategy to** provide quick, convenient banking

services, anytime, anywhere. (**Platinum Edge addition**)

As you can see, a vision statement identifies a future state for the product when the product reaches completion. The vision focuses on the conditions that should exist when the product is complete.



**WARNING** Avoid generalizations in your vision statement such as “make customers happy” or “sell more products.” Also watch out for too much technological specificity, such as “using release 9.x of Java, create a program with four modules that ...” At this early stage, defining specific technologies might limit you later.

Here are a few extracts from vision statements that should ring warning bells:

- » Secure additional customers for the MyXYZ application.
- » Satisfy our customers by December.
- » Eliminate all defects and improve quality.
- » Create a new application in Java.
- » Beat the Widget Company to market by six months.

### ***Step 3: Validating and revising the vision statement***

After you draft your vision statement, review it against the following quality checklist:

- » Is this vision statement clear, focused, and written for an internal audience?
- » Does the statement provide a compelling description of how the product meets customer needs?
- » Does the vision describe the best possible outcome?
- » Is the business objective specific enough that the goal is achievable?
- » Does the statement deliver value that is consistent with corporate strategies and goals?
- » Is the product vision statement compelling?

» Is the vision concise?

These yes-or-no questions will help you determine whether your vision statement is thorough and clear. If any answers are no, revise the vision statement.

When all answers are yes, move on to reviewing the statement with others, including the following:

- » **Project stakeholders:** The stakeholders will be able to identify that the vision statement includes everything the product should accomplish.
- » **Your development team:** The team, because it will create the product, must understand what the product needs to accomplish.
- » **Scrum master:** A strong understanding of the product will help the scrum master remove roadblocks and ensure that the development team is on the right path later in the project.
- » **Agile mentor:** Share the vision statement with your agile mentor, if you have one. The agile mentor is independent of the organization and can provide an external perspective, qualities that can make for a great objective voice.

See whether others think the vision statement is clear and delivers the message you want to convey. Review and revise the vision statement until the project stakeholders, the development team, and the scrum master fully understand the statement.



**REMEMBER** At this stage of your project, you might not have a development team or scrum master. After you form a scrum team, be sure to review the vision statement with it.

## ***Step 4: Finalizing the vision statement***

After you finish revising the vision statement, make sure your development team, scrum master, and stakeholders have the final copy. You might even put a copy on the wall in the scrum team's work area, where you can see it every day. You will refer to the vision statement throughout the life of the

project.

If your project is more than a year long, you may want to revisit the vision statement. We like to review the product vision statement at least once a year to make sure the product reflects the marketplace and supports any changes in the company's needs. Because the vision statement is the long-term boundary of the project, the project should end when the vision is no longer viable.



**REMEMBER** The product owner owns the product vision statement and is responsible for its preparation and communication across the organization. The product vision sets expectations for stakeholders and helps the development team stay focused on the goal.

Congratulations. You've just completed the first stage in your agile project. Now it's time to create a product roadmap.

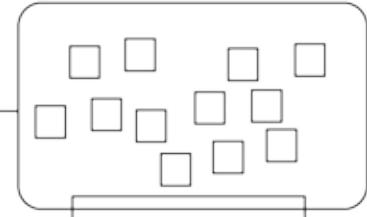
## ***Creating a Product Roadmap***

The product roadmap, stage 2 in the Roadmap to Value (see [Figure 7-4](#)), is an overall view of the product's requirements and a valuable tool for planning and organizing the journey of product development. Use the product roadmap to categorize requirements, prioritize them, identify gaps and dependencies, and determine a timetable for releasing to the customer.

A common agile practice

### **Stage 2: PRODUCT ROADMAP**

**Description:** Holistic view of product features that create the product vision  
**Owner:** Product owner  
**Frequency:** At least biannually



**FIGURE 7-4:** The product roadmap as part of the Roadmap to Value.

As he or she does with the product vision statement, the product owner creates the product roadmap, with help from the development team and

stakeholders. The development team participates to a greater degree than it did during the creation of the vision statement.



TIP Keep in mind that you will refine requirements and effort estimates throughout the project. In the product roadmap phase, it's okay for your requirements, estimates, and time frames to be at a very high level.

To create your product roadmap, you do the following:

- 1. Identify stakeholders.**
- 2. Establish product requirements and add them to the roadmap.**
- 3. Arrange the product requirements based on values, risks, and dependencies.**
- 4. Estimate the development effort at a high level and prioritize the product's requirements.**
- 5. Determine high-level time frames for releasing groups of functionality to the customer.**

Because priorities can change, expect to update your product roadmap throughout the project. We like to update the product roadmap at least twice a year.



TIP Your product roadmap can be as simple as sticky notes arranged on a whiteboard, which makes updates as easy as moving a sticky note from one section of the whiteboard to another.

You use the product roadmap to plan releases — stage 3 in the Roadmap to Value. *Releases* are groups of usable product functionality that you release to customers to gather real-world feedback and to generate return on investment.

The following section details the steps to create a product roadmap.

## ***Step 1: Identifying stakeholders***

When initially establishing the product vision, it's likely you will have

identified only a few key stakeholders who are available to provide high-level feedback. At the product roadmap stage, you put more context to the product vision and identify how you achieve the vision, which gives more insight into who will have a stake in your project.

This is the time to engage with existing and newly identified stakeholders to gather feedback on the functionality you want to implement to achieve the vision. The product roadmap is your first cut at a high-level product backlog, discussed later in this chapter. With this first round of detail identified, you'll want to engage more than just the scrum team, project sponsor, and obvious users. Consider including the following people:

- » **Marketing department:** Your customers need to know about your product, and that's what the marketing department provides. They need to understand your plans, and may have input into the order in which you release functionality to the market, based on their experience and research.
- » **Customer service department:** Once your product is in the market, how will it be supported? Specific roadmap items might identify the person you'll need to prepare for support. For instance, a product owner may not see much value in plugging in a live online chat feature, but a customer service manager may see it differently because his or her representatives can handle simultaneously only one phone call but as many as six chat sessions.
- » **Sales department:** Make sure that the sales team members see the product so that they start selling the same thing you are building. Like the marketing department, the sales department will have first-hand knowledge about what your customers are looking for.
- » **Legal department:** Especially if you're in a highly regulated industry, review your roadmap with legal counsel as early as possible to make sure you haven't missed anything that could put your project at risk if discovered later in the project.
- » **Additional customers:** While identifying features on your roadmap, you may discover additional people who will find value in what you will create. Give them a chance to review your roadmap to validate your assumptions.

## **Step 2: Establishing product requirements**

The second step in creating a product roadmap is to identify, or define, the different requirements for your product.

When you first create your product roadmap, you typically start with large, high-level requirements. The requirements on your product roadmap will most likely be at two different levels: themes and features. *Themes* are logical groups of features and requirements at their highest levels. *Features* are parts of the product at a very high level and describe a new capability the customer will have once the feature is complete.

### **DECOMPOSING REQUIREMENTS**

Throughout the project, you'll break down requirements into smaller, more manageable parts using a process called *decomposition*, or *progressive elaboration*. You can break down requirements into the following sizes, listed from largest to smallest:

**Themes:** A *theme* is a logical group of features and is also a requirement at its highest level. You may group features into themes in your product roadmap.

**Features:** *Features* are parts of products at a very high level. Features describe a new capability the customers will have once the feature is complete. You use features in your product roadmap.

**Epic user stories:** *Epics* are medium-sized requirements that are decomposed from a feature and often contain multiple actions or channels of value. You need to break down your epics before you can start creating functionality from them. You can find out how you use epics for release planning in [Chapter 8](#).

**User stories:** *User stories* are requirements that contain a single action or integration and are small enough to start implementing into functionality. You see how you define user stories and use them at the release and sprint level in [Chapter 8](#).

**Tasks:** *Tasks* are the execution steps required to develop a requirement into working functionality. You break down user stories into different tasks during sprint planning. You can find out about tasks and sprint planning in [Chapter 8](#).

Keep in mind that each requirement may not go through all these sizes. For example, you may create a particular requirement at the user story level, and never think of it on the theme or epic scale. You may create a requirement at the epic user story level, but it may be a lower-priority requirement. Because of just-in-time planning, you may not take the time to decompose that lower-priority epic user story until you complete development of all the higher-priority requirements.

To identify product themes and features, the product owner can work with stakeholders and the development team. It may help to have a requirements

session, where the stakeholders and the development team meet and write down as many requirements that they can think of.



TIP When you start creating requirements at the theme and feature level, it can help to write those requirements on index cards or big sticky notes. Using a physical card that you can move from one category to another and back again can make organizing and prioritizing those requirements very easy.

While you create the product roadmap, the features you identify start to make up your *product backlog* — the full list of what is in scope for a product, regardless of level of detail. Once you have identified your first product features, you have your product backlog started.

### ***Step 3: Arranging product features***

After you identify your product features, you work with the stakeholders to group them into *themes* — common, logical groups of features. A stakeholder meeting works well for grouping features, just like it works for creating requirements. You can group features by usage flow, technical similarity, or business need.

Visualizing themes and features on your roadmap allows you to assign business value and risks associated with each feature relative to others. The product owner, along with the development team and stakeholders, can also identify dependencies between features, locate any gaps, and prioritize the order in which each feature should be developed based on each of these factors.

Here are questions to consider when grouping and ordering your requirements:

- » How would customers use our product?
- » If we offered this proposed feature, what else would customers need to do? What else might they want to do?
- » Can the development team identify technical affinities or dependencies?

Use the answers to these questions to identify your themes. Then group the

features by these themes. For example, in the mobile banking application, the themes might be

- » Account information
- » Transactions
- » Customer service functions
- » Mobile functions

[Figure 7-5](#) shows features grouped by themes.

Common activities	Reduction in call volume
Authenticate and access my accounts	Pay bills
View balance	Order checks
View pending transactions	Transfer money between accounts
View bills	Put a stop on a check or range of checks
Find a branch/ATM machine	Open an account
Call customer service	Change password

[FIGURE 7-5:](#) Features grouped by themes.

## ***Step 4: Estimating efforts and ordering requirements***

You've identified your product requirements and arranged those requirements

into logical groups. Next, you estimate and prioritize the requirements. Here are a few terms you need to be familiar with:

- » *Effort* is the ease or difficulty of creating functionality from a particular requirement.
- » An *estimate*, as a noun, can be the number or description you use to express the estimated effort of a requirement.
- » *Estimating* a requirement, as a verb, means to come up with an approximate idea of how easy or hard (how much effort) that requirement will be to create.
- » *Ordering*, or *prioritizing*, a requirement means to determine that requirement's value and risk in relation to other requirements, and in what order you will implement them.
- » *Value* means how beneficial a product requirement might be to the organization creating that product.
- » *Risk* refers to the negative effect a requirement can have on the project.



TIP You can estimate and prioritize requirements at any level, from themes and features down to single user stories.

Prioritizing requirements is really about ordering them. You can find various methods — many of them complicated — for determining the priority of product backlog items. We keep things simple by creating an ordered to-do list of product backlog items, based on business value, risk, and effort, listed in the order in which you will implement them. Forcing an order requires making a priority decision for every requirement relative to every other requirement. A scrum team can work on one thing at a time, so it is important to format your product roadmap accordingly.

To score your requirements, you work with two different groups of people:

- » The development team determines the effort to implement the functionality for each requirement.
- » The product owner, with support from the stakeholders, determines the

value and risk of the requirement to the customer and the business.

## ***Estimating effort***

To order requirements, the development team must first estimate the effort for each requirement relative to all other requirements.

In [Chapter 8](#), we show you relative estimation techniques that agile teams use to accurately estimate effort. Traditional estimation methods aim for precision by using absolute time estimates at every level of the project schedule, whether the team is working on the work items today or two years from now. This practice gives non-agile teams a false sense of precision and isn't accurate in reality (as thousands of failed projects prove). How could you possibly know what each team member will be working on six months from now, and how long it will take to do that work, when you are just starting to learn about the project at the beginning?

*Relative estimating* is a self-correcting mechanism that allows agile teams to be more accurate because it's much easier to be right when comparing one requirement against another and determining whether one is bigger than another, and by roughly how much.

To order your requirements, you also want to know any dependencies. Dependencies mean that one requirement is a predecessor for another requirement. For example, if you were to have an application that needs someone to log in with a username and password, the requirement for creating the username would be a dependency for the requirement for creating the password, because you generally need a username to set up a password.

## ***Assessing business value and risk***

Together with stakeholders, the product owner identifies the highest business value items (either high potential ROI or other perceived value to the end customer), as well as those items with high negative impact on the project if unresolved.

Similar to effort estimates, values or risks can be assigned to each product roadmap item. For example, you might assign value using monetary ROI amounts or, for an internally used product, assign value or risk by using high, medium, or low.

Effort, business value, and risk estimates inform the product owner's prioritization decisions for each requirement. The highest value and risk items should be at the top of the product roadmap. High-risk items should be explored and implemented first to avoid rear-loading the project's risk. If a high-risk item will cause a project to fail (an issue that cannot be resolved), agile teams learn about it early. If a project is going to fail, you want to fail early, fail cheap, and move on to a new project that has value. In that sense, failure is a form of success for an agile team.

After you have your value, risk, and effort estimates, you can determine the relative priority, or order, of each requirement.

- » A requirement with high value or high risk (or both) and low effort will have a high relative priority. The product owner might order this item at the top of the roadmap.
- » A requirement with low value or low risk (or both) and high effort will have a lower relative priority. This item will likely end up toward the bottom of the roadmap.



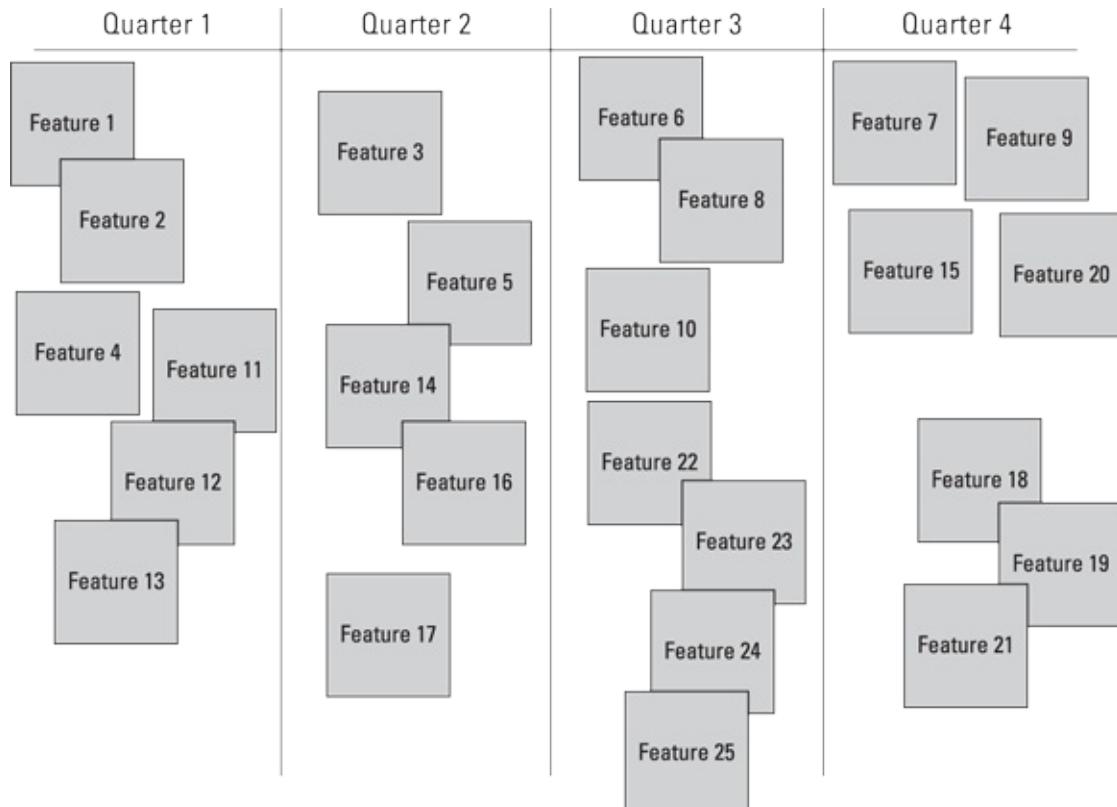
WARNING Relative priority is only a tool to help the product owner make decisions and prioritize requirements. It isn't a mathematical universal that you must follow. Make sure your tools help rather than hinder.

## **Prioritizing requirements**

To determine the overall priority for your requirements, answer the following questions:

- » What is the relative priority of the requirement?
- » What are the prerequisites for any requirement?
- » What set of requirements belong together and will constitute a solid set of functionality you can release to the customer?

Using the answers to these questions, you can place the highest-priority requirements first in the product roadmap. When you've finished prioritizing your requirements, you'll have something that looks like [Figure 7-6](#).



**FIGURE 7-6:** Product roadmap with ordered requirements.

Your prioritized list of requirements is called a *product backlog*. Your product backlog is an important agile document, or *artifact*. You use this backlog throughout your entire project.

With a product backlog in hand, you can start adding target releases to your product roadmap.

## Step 5: Determining high-level time frames

When you create your product roadmap, your time frames for releasing product requirements are at a very high level. For the initial roadmap, choose a logical time increment for your project, such as a certain number of days, weeks, months, quarters (three-month periods), or even larger increments. Using both the requirement and the priority, you can add requirements to each increment of time.



**REMEMBER** Creating a product roadmap might seem like a lot of work, but after you get the hang of it, you can create one in a short time. Some scrum

teams can create a product vision, a product roadmap, and a release plan and be ready to start their sprint in as little as one day! To begin developing the product, you need only enough requirements for your first sprint. You can determine the rest as the project progresses.

## **Saving your work**

Up until now, you could do all your roadmap planning with whiteboards and sticky notes. After your first full draft is complete, however, save the product roadmap, especially if you need to share the roadmap with remote stakeholders or development team members. You could take a photo of your sticky notes and whiteboard, or you could type the information into a document and save it electronically.

You update the product roadmap throughout the project, as priorities change. For now, the contents of the first release should be clear — and that's all you need to worry about at this stage.

# **Completing the Product Backlog**

The product roadmap contains high level features and some tentative release timelines. The requirements on your product roadmap are the first version of your *product backlog*.

The product backlog is the list of all requirements associated with the project. The product owner is responsible for creating and maintaining the product backlog by adding and prioritizing requirements. The scrum team uses the prioritized product backlog throughout the project to plan its work — like a streamlined project plan.

[Figure 7-7](#) shows a sample product backlog. At a minimum, when creating your product backlog, be sure to do the following:

- » Include a description of each requirement.
- » Order the requirements based on priority.
- » Add the effort estimate.

## PRODUCT BACKLOG

Order	ID	Item	Type	Status	Estimate
1	121	As an Administrator, I want to link accounts to profiles, so that customers can access new accounts.	Requirement	Not Started	5
2	113	Update requirements traceability matrix.	Overhead	Not Started	2
3	403	Test automation training for Michael.	Improvement	Not Started	3
4	97	Refactor Login Class.	Maintenance	Not Started	8
5	68	As a Site Visitor, I want to find locations, so that I can use bank services.	Requirement	Not Started	8

**FIGURE 7-7:** Product backlog items sample.

We also like to include the type of backlog item as well as the status. Scrum teams will work mainly on developing features as described in the words of the user (user stories). But there may be need for other types of product backlog items, such as overhead items (things the scrum team determines are needed but don't contribute to the functionality), maintenance items (design improvements that need to be done to the product or system but don't directly increase value to the customer), or improvement items (action items for process improvements identified in the sprint retrospective). You can see examples of each of these in [Figure 7-7](#).



**REMEMBER** In [Chapter 2](#), we explain how documents for agile projects should be barely sufficient, with only information that is absolutely necessary to create the product. If you keep your product backlog format simple and barely sufficient, you'll save time updating it throughout the project.

The scrum team refers to the product backlog as the main source for project requirements. If a requirement exists, it's in the product backlog. The requirements in your product backlog will change throughout the project in several ways. For example, as the team completes requirements, you mark those requirements as complete in the product backlog. You also record any new requirements gathered based on feedback from stakeholders and customers. Some requirements will be updated with new or clarified information, broken down into smaller user stories, or refined in other ways. Additionally, you update the priority and effort scores of existing

requirements as needed.

The total number of story points in the product backlog — all user story points added together — is your current *product backlog estimate*. This estimate changes daily as user stories are completed and new user stories are added. Discover more about using the product backlog estimate to predict the project length and cost in [Chapter 13](#).



**REMEMBER** Keep your product backlog up to date so that you always have accurate cost and schedule estimates. A current product backlog also gives you the flexibility to prioritize newly identified product requirements — a key agile benefit — against existing features.

After you have a product backlog, you can begin planning releases and sprints, which we show you in the next chapter.

# Chapter 8

## Planning Releases and Sprints

---

### IN THIS CHAPTER

- » Decomposing requirements and creating user stories
- » Creating a product backlog, release plan, and sprint backlog
- » Planning sprints

After you create a product roadmap for your agile project (see [Chapter 7](#)), it's time to start elaborating on your product details. In this chapter, you discover how to break down your requirements to a more granular level, refine your product backlog, create a release plan, and build a sprint backlog for execution. First, you see how to break down the larger requirements from your product roadmap into smaller, more manageable requirements called *user stories*.



REMEMBER The concept of breaking down requirements into smaller pieces is called *decomposition*.

### ***Refining Requirements and Estimates***

You start agile projects with very large requirements. As the project progresses and you get closer to developing those requirements, you will break them down into smaller parts — small enough to begin developing.

One clear, effective format for defining product requirements is the user story. The user story and its larger cousin, the epic user story, are good-sized requirements for release planning and sprint planning. In this section, you find out how to create a user story, prioritize user stories, and estimate user story effort.

#### ***What is a user story?***

The *user story* is a simple description of a product requirement in terms of what that requirement must accomplish for whom. Traditional requirements usually read something like this: “The system shall [insert technical description].” This requirement addresses only the technical nature of what will be done; the overall business objective is unclear. Because the development team has the context to engage more deeply, it clearly knows the benefit to the user (or the customer or the business) of each requirement and delivers what the customer wants faster and with higher quality.

Your user story will have, at a minimum, the following parts:

Title (recognizable name for the user story)

As a (type of user)

I want to (take this action)

so that (I get this benefit)

The user story also includes a list of validation steps (*acceptance criteria*) to take so you know that the working requirement for the user story is correct:

When I (take this action), (this happens)

User stories may also include the following:

- » **A user story ID:** A number to differentiate this user story from other user stories.
- » **The user story value and effort estimate:** *Value* is how beneficial a user story might be to the organization creating that product. *Effort* is the ease or difficulty in creating that user story. We introduce how to score a user story’s business value, risk, and effort in [Chapter 7](#).
- » **The name of the person who thought of the user story:** Anyone on the project team can create a user story.



TIP Although agile project management approaches encourage low-tech tools, the scrum team should also find out what works best for it in each

situation. A lot of electronic user story tools are available, some of which are free. Some are simple and are only for user stories. Others are complex and will integrate with other product documents. We love index cards, but that solution may not be for everyone. Use what works best for your scrum team and your project.

[Figure 8-1](#) shows a typical user story card, front and back. The front has the main description of the user story. The back shows how you will confirm that the requirement works correctly, after the development team has created the functionality.

Title Transfer money between accounts	When I do this:	This happens:
As Carol,	When I view my account balances,	I see an option to transfer funds.
I want to transfer funds between accounts	When I select the transfer option,	I choose between which accounts I want to transfer funds.
so that each account has the correct amount of funds	When I select the "transfer from" option,	I see a list of my available accounts and balances.
Value _____	When I select the "transfer to" option,	I see a list of my available accounts and balances.
Jennifer Author		
Estimate _____		

[FIGURE 8-1:](#) Card-based user story example.

The product owner gathers the user stories and manages them (that is, determines the priority and initiates the decomposition discussions). The development team and other stakeholders are also involved in creating and decomposing user stories.



TIP Note that user stories aren't the only way to describe product requirements. You could simply make a list of requirements without any given structure. However, because user stories include a lot of useful information in a simple, compact format, we find that they are very effective in conveying exactly what a requirement needs to do for the customer.

The big benefit of the user story format is when the development team starts to create and test requirements. The development team members know exactly for whom they are creating the requirement, what the requirement should do, and how to double-check that the requirement satisfies the

intention of the requirement.

We use user stories as examples of requirements throughout the chapter and the book. Keep in mind that anything we describe that you can do with user stories, you can do also with more generically expressed requirements.

## ***Steps to create a user story***

When creating a user story, follow these steps:

- 1. Identify the project stakeholders.**
- 2. Identify who will use the product.**
- 3. Working with the stakeholders, write down the requirements that the product will need and use the format described earlier to create your user stories.**

Find out how to follow these three steps in the following sections.



**REMEMBER** Being agile and adaptive requires iterating. Don't spend a ton of time trying to identify every single requirement your product might have.

You can always add requirements later in the project. The best changes often come at the end of a project, when you know the most about the product and the customers.

### ***Identifying project stakeholders***

You probably have a good idea about who your project stakeholders are — anyone involved with, affected by, or who can affect the product and its creation.



**REMEMBER** You will also work with stakeholders when you create your product vision and your product roadmap.

Make sure the stakeholders are available to help you create requirements. Stakeholders of the sample mobile banking application introduced in [Chapter 7](#) might include the following:

- » People who interact with customers on a regular basis, such as customer service representatives or bank branch personnel.
- » Business experts for the different areas where your product's customers interact. For example, XYZ Bank might have one manager in charge of checking accounts, another manager in charge of savings accounts, and a third manager in charge of online bill payment services. If you're creating a mobile banking application, all these people would be project stakeholders.
- » Users of your product, if they're available.
- » Experts of the type of product you're creating. For example, a developer who has created mobile applications, a marketing manager who knows how to create mobile campaigns, and a user experience specialist who specializes in mobile interfaces all might be helpful on the sample XYZ Bank mobile banking project.
- » Technical stakeholders. These are people who work with the systems that might need to interact with your product.

## ***Identifying users***

Your customers and stakeholders provide requirements for the product owner to vet for placement on your product backlog. Your customers may or may not be the same people who will use your product. Knowing who your end users are and how they will interact with your product drive how you define and implement each requirement on your product roadmap.

With your product roadmap visualized, you can identify each type of user. For the mobile banking application, you would have individual and business bankers. The individual category would include youth, young adults, students, and single, married, retired, and wealthy users. Businesses of all sizes might be represented. Employee users would include tellers, branch managers, account managers, and fund managers. Each type of user will interact with your application in different ways and for different reasons. Knowing who these people are enables you to better define the purpose and desired benefits of each of their interactions.

We like to define users using *personas*, or a written description about a type of user represented by a fictitious person. For instance, “Robert is a 65-year-old retired engineer who is spending his retirement traveling the world. His

net worth is \$1,000,000, and he has residual income from several investment real estate properties.”

“Robert” represents 30 percent of XYZ Bank’s customers, and a good portion of the product roadmap includes features that someone like Robert will use. Instead of repeating all the details about Robert every time the scrum team discusses these features, they can simply refer to the type of user as “Robert.” The product owner might identify several of these, as needed, and will even print the descriptions with a stock photo of what Robert might look like and post them on the wall in the team’s work area to refer to throughout the project.



**TIP** Know who your users are, so you can develop features they’ll actually use.

Suppose that you’re the product owner for the XYZ Bank’s mobile banking project. You’re responsible for the department that will bring the product to market, preferably in the next six months. You have the following ideas about the application’s users:

- » The customers (the end users of the application) probably want quick access to up-to-date information about their balances and recent transactions.
- » Maybe the customers are about to buy a large-ticket item, and they want to make sure they can charge it.
- » Maybe the customers’ ATM cards were just refused, but they have no idea why, and they want to check recent transactions for possible fraudulent activities.
- » Maybe the customers just realized that they forgot to pay their credit card bill and will have penalty charges if they don’t pay the card today.

Who are your personas for this application? Here are a few examples:

- » **Persona #1:** Jason is a young, tech-savvy executive who travels a lot. When he has a spare moment, he wants to handle personal business quickly. He carefully invests his money in high-interest portfolios. He

keeps his available cash low.

- » **Persona #2:** Carol is a small-business owner who stages properties when clients are trying to sell their home. She shops at consignment centers and often finds furnishings she wants to buy for her clients.
- » **Persona #3:** Nick is a student who lives on student loans and a part-time job. He knows he can be flaky with money because he's flaky with everything else. He just lost his checkbook.



TIP Your product stakeholders can help you create personas. Find people who are experts on the day-to-day business for your product. Those stakeholders will know a lot about your potential customers.

### *Determining product requirements and creating user stories*

After you have identified your different customers, you can start to determine product requirements and create user stories for the personas. A good way to create user stories is to bring your stakeholders together for a user story creation session.

Have the stakeholders write down as many requirements as they can think of, using the user story format. One user story for the project and personas from the preceding sections might be as follows:

- » Front side of card:

- **Title** See bank account balance
- **As** Jason,
- **I want to** see my checking account balance on my smartphone
- **so that** I can see how much money I have in my checking account

- » Back side of card:

- **When I** sign into the XYZ Bank mobile application, my checking account balance appears at the top of the page.
- **When I** sign into the XYZ Bank mobile application after making a purchase or a deposit, my checking account balance reflects that purchase or deposit.

You can see sample user stories in card format in [Figure 8-2](#).

<p><b>Title</b> Transfer money between accounts</p> <p><b>As</b> Carol,</p> <p><b>I want to</b> categorize expenses, <b>so that</b> I can easily identify my purchases made for my clients.</p>		
Value	Jennifer	Estimate
Author		

<p><b>Title</b> Put stop on a check</p> <p><b>As</b> Nick,</p> <p><b>I want to</b> stop payment on a lost or stolen check, <b>so that</b> I can avoid any unauthorized activity on my account.</p>		
Value	Caroline	Estimate
Author		

[FIGURE 8-2:](#) Sample user stories.



**REMEMBER** Be sure to continuously add and prioritize new user stories to your product backlog. Keeping your product backlog up-to-date will help you

have the highest-priority user stories when it is time to plan your sprint.

Throughout an agile project, you will create new user stories. You'll also take existing large requirements and decompose them until they're manageable enough to work on during a sprint.

## ***Breaking down requirements***

You refine requirements many times throughout an agile project. For example:

- » When you create the product roadmap (see [Chapter 7](#)), you create features (capabilities your customers will have after you develop the features), as well as themes (logical groups of features). Although features are intentionally large, we require features at the product roadmap level to be no larger than 144 story points on the Fibonacci scale.
- » When you plan releases, you break down the features into more concise user stories. User stories at the release plan level can be either *epics*, very large user stories with multiple actions, or individual user stories, which contain a single action. For our clients, user stories at the release plan level should be no larger than 34 story points. You find out more about releases later in this chapter.
- » When you plan sprints, you can break down user stories even further. You also identify individual tasks associated with each user story in the sprint. For our clients, user stories at the sprint level should be no larger than eight story points. Tasks will be estimated in hours and should be no larger than what can be accomplished in a day.

To decompose requirements, you'll want to think about how to break down the requirement into individual actions. [Table 8-1](#) shows a requirement from the XYZ Bank application introduced in [Chapter 7](#) that is decomposed from the theme level down to the user story level.

**TABLE 8-1 Decomposing a Requirement**

<b><i>Requirement Level</i></b>	<b><i>Requirement</i></b>
Theme	See bank account data on a mobile device.
	See account balances.
	See a list of recent withdrawals or purchases.
	See a list of recent deposits.

Features	<p><a href="#">See a list of recent deposits.</a></p> <p><a href="#">See my upcoming automatic bill payments.</a></p> <p><a href="#">See my account alerts.</a></p>
Epic user stories — decomposed from “see account balances”	<p><a href="#">See checking account balance.</a></p> <p><a href="#">See savings account balance.</a></p> <p><a href="#">See loan balance.</a></p> <p><a href="#">See investment account balance.</a></p> <p><a href="#">See retirement account balance.</a></p>
User stories — decomposed from “see checking account balance”	<p><a href="#">See a list of my accounts once securely logged in.</a></p> <p><a href="#">Select and view my checking account.</a></p> <p><a href="#">See account balance changes after withdrawals.</a></p> <p><a href="#">See account balance changes after purchases.</a></p> <p><a href="#">See day's end account balance.</a></p> <p><a href="#">See available account balance.</a></p> <p><a href="#">See mobile application navigation items.</a></p> <p><a href="#">Change account view.</a></p> <p><a href="#">Log out of mobile application.</a></p>

## USER STORIES AND THE INVEST APPROACH

You may be asking, just how decomposed does a user story have to be? Bill Wake, in his blog at XP123.com, describes the INVEST approach to ensure quality in user stories. We like his method so much we include it here.

Using the INVEST approach, user stories should be

- **Independent:** To the extent possible, a story should need no other stories to implement the feature that the story describes.
- **Negotiable:** Not overly detailed. The user story has room for discussion and an expansion of details.
- **Valuable:** The story demonstrates product value to the customer. The story describes features, not technical tasks to implement it. The story is in the user's language and is easy to explain. The people using the product or system can understand the story.
- **Estimable:** The story is descriptive, accurate, and concise, so the developers can generally estimate the work necessary to create the functionality in the user story.
- **Small:** It is easier to plan and accurately estimate small user stories. A good rule of thumb is that the development team can complete 6-10 user stories in a sprint.
- **Testable:** You can easily validate the user story, and the results are definitive.

## **Estimation poker**

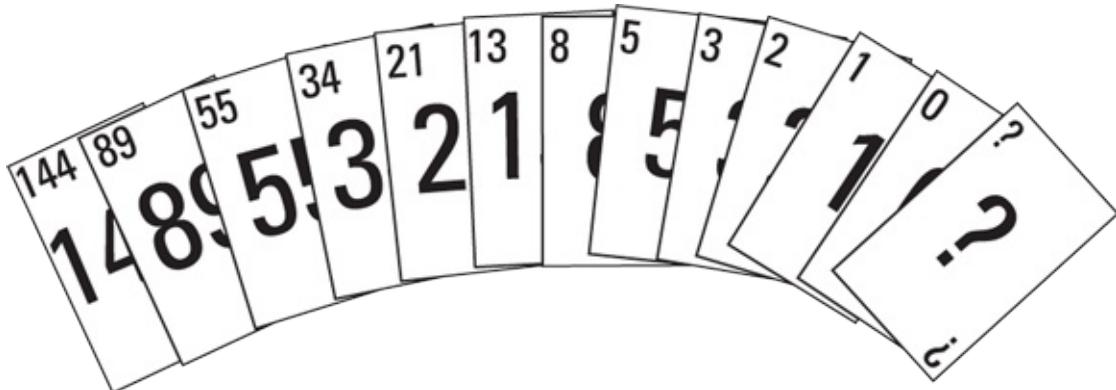
As you refine your requirements, you need to refine your estimates as well. It's time to have some fun!

One of the most popular ways of estimating user stories is by playing *estimation poker*, sometimes called *planning poker*, a game to determine user story size and to build consensus among the development team members.



**REMEMBER** The scrum master can help coordinate estimation, and the product owner can provide information about features, but the development team is responsible for estimating the level of effort required for the user stories. After all, the development team has to do the work to create the features that those stories describe.

To play estimation poker, you need a deck of cards like the one in [Figure 8-3](#). You can get a digital version online at our website ([www.platinumedge.com/estimationpoker](http://www.platinumedge.com/estimationpoker)), or you can make your own with index cards and markers. The numbers on the cards are from the Fibonacci sequence.



**FIGURE 8-3:** A deck of estimation poker cards.

The Fibonacci sequence follows this progression:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, and so on

Each number after the first two is the sum of the previous two numbers.

Each user story receives an estimate relative to other user stories. For

instance, a user story that is a 5 requires more work than a 3, a 2, and a 1. It is about 5 times as much effort as a 1, more than double the effort of a 2, and roughly the amount of effort as a 3 and a 2 combined. It is not as much effort as an 8, but is just over half the effort of an 8.

As user stories and epic user stories increase in size, the difference between Fibonacci numbers gets bigger. Acknowledging these increasing gaps in precision for larger requirements is why the Fibonacci sequence works so well for relative estimation.

To play estimation poker, follow these steps:

- 1. Provide each member of the development team with a deck of estimation poker cards.**
- 2. From the list of user stories presented by the product owner, the team agrees on one user story that would be a 5.**

The team follows two rules: (1) The development team should not allow any single user story larger than an 8 to be pulled into a sprint, and (2) scrum teams should be able to complete roughly 6-10 user stories in a sprint.

The scrum master helps the development team reach consensus by using fist of five or thumbs up/thumbs down (as described in [Chapter 6](#)). This user story becomes the *anchor story*.

- 3. The product owner reads a high-priority user story to the players.**
- 4. Each player selects a card representing his or her estimate of the effort involved in the user story and lays the card facedown on the table.**

The players should compare the user story to other user stories they have estimated. (The first time through, the players compare the user story to only the anchor story.) Make sure no other players can see your card.

- 5. All players turn over their cards simultaneously.**
- 6. If the players have different story points:**

- a. It's time for discussion.**

The players with the highest and lowest scores talk about their assumptions and why they think the estimate for the user story should be higher or lower, respectively. The players compare the

- effort for the user story against the anchor story. The product owner provides more clarification about the story, as necessary.
- b. *Once everyone agrees on assumptions and has any necessary clarifications, the players reevaluate their estimates and place their new selected cards on the table.*
  - c. *If the story points are different, the players repeat the process, usually up to three times.*
  - d. *If the players can't agree on the estimated effort, the scrum master helps the development team determine a score that all the players can support (he or she may use fist of five or thumbs up/thumbs down, as described in [Chapter 6](#)), or determine that the user story requires more detail or needs to be further broken down.*

## 7. The players repeat Steps 3 through 6 for each user story.



**REMEMBER** Consider each part of the definition of *done* — developed, integrated, tested, and documented — when you create estimates.

You can play estimation poker at any point — but definitely play during the product roadmap development and as you progressively break down user stories for inclusion in releases and sprints. With practice, the development team will get into a planning rhythm and become more adept at quickly estimating.



**TIP** On average, development teams will spend about 10 percent of their time on a project decomposing requirements, including estimating and reestimating. Make your estimation poker games fun! Bring in snacks, take breaks as needed, use humor, and keep the mood light.

## **Affinity estimating**

Estimation poker can be effective, but what if you have many user stories? Playing estimation poker for, say, 500 user stories could take a long time. You need a way to focus on only the user stories you must discuss to gain

consensus.

When you have a large number of user stories, many of them are probably similar and would require a similar amount of effort to complete. One way to determine the right stories for discussion is to use affinity estimating. In *affinity estimating*, you quickly categorize your user stories and then apply estimates to these categories of stories.



**TIP** When estimating by affinity, write your user stories on index cards or sticky notes. These types of user story cards work well when quickly categorizing stories.

Affinity estimating can be a fast and furious activity — the development team may choose to have the scrum master help facilitate affinity estimating sessions. To estimate by affinity, follow these steps:

- 1. Taking no more than 60 seconds for each category, the development team agrees on a single user story in each of the following categories:**

- Extra-small user story
- Small user story
- Medium user story
- Large user story
- Extra-large user story
- Epic user story that is too large to come into the sprint
- Needs clarification before estimating

- 2. Taking no more than 60 seconds per user story, the development team puts all remaining stories into the categories listed in Step 1.**

If you're using index cards or sticky notes for your user stories, you can physically place those cards into categories on a table or a whiteboard, respectively. If you split the user stories among the development team members, having each development team member categorize a group of stories, this step can go quickly!

- 3. Taking another 30 minutes, maximum, for each 100 stories, the**

## **development team reviews and adjusts the placement of the user stories.**

The entire development team must agree on the placement of the user stories into size categories.

- 4. The product owner reviews the categorization.**
- 5. When the product owner's expected estimate and the team's actual estimate differ by more than one story size, they discuss that user story.**

The development team may or may not decide to adjust the story size.
- 6. The development team plays estimation poker on the user stories in both the epic and the needs clarification categories.**

The number of user stories in these categories should be minimal.



**REMEMBER** Note that after the product owner and the development team discuss clarifications, the development team has the final say on the user story size.

User stories in the same size category will have the same user story score. You can play a round of estimation poker to double-check a few, but you won't need to waste time in unnecessary discussion for every user story.

Story sizes are like T-shirt sizes and should correspond to Fibonacci scale numbers, as shown in [Figure 8-4](#).

SIZE	POINTS
Extra small (XS)	1
Small (S)	2
Medium (M)	3
Large (L)	5
Extra large (XL)	8

**FIGURE 8-4:** Story sizes as T-shirt sizes and their Fibonacci numbers.



**TIP** You can use the estimating and prioritizing techniques in this chapter for requirements at any level, from themes and features down to single user stories.

That's it. In a few hours, your entire product backlog was estimated. In addition, your scrum team has a shared understanding of what the requirements mean, having discussed them face to face rather than relying on interpretations of extensive documentation.

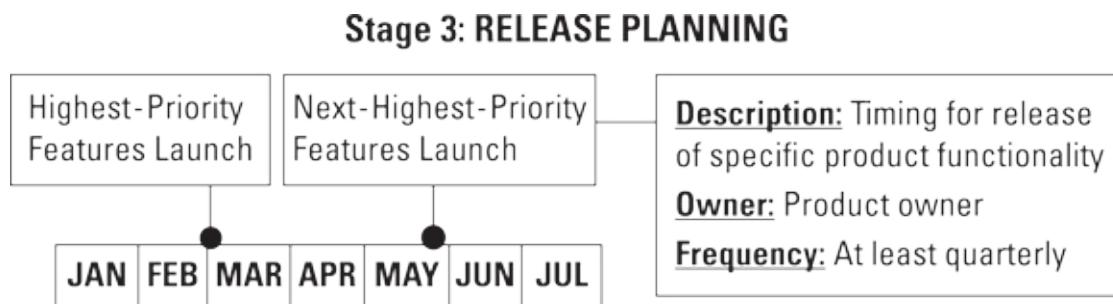
## Release Planning

A *release* is a group of usable product features that you deploy to the market. A release does not need to include all the functionality outlined in the product roadmap but should include at least the *minimal marketable features*, the

smallest group of product features that you can effectively deploy and promote in the marketplace. Your early releases will exclude many of the medium-and low-priority requirements you identified during the product roadmap stage.

When planning a release, you establish the next set of minimal marketable features and identify an imminent product launch date around which the team can mobilize. As when creating the vision statement and the product roadmap, the product owner is responsible for creating the release goal and establishing the release date. However, the development team's estimates, with the scrum master's facilitation, contribute to the process.

Release planning is stage 3 in the Roadmap to Value (refer to [Chapter 7](#) to see the roadmap as a whole). [Figure 8-5](#) shows how release planning fits into an agile project.



(Stages 1-3 are common practices outside of scrum)

**FIGURE 8-5:** Release planning as part of the Roadmap to Value.

Release planning involves completing two key activities:

- » **Revising the product backlog:** In [Chapter 7](#), we tell you that the product backlog is a comprehensive list of all the user stories you currently know for your project, whether or not they belong in the current release. Keep in mind that your list of user stories will probably change throughout the project.
- » **Creating the release plan:** This activity consists of the release goal, release target date, and prioritization of product backlog items that support the release goal. The release plan provides a midrange goal that the team can accomplish.



WARNING Don't create a new, separate backlog during release planning. The task is unnecessary and reduces the product owner's flexibility.

Prioritizing the existing product backlog based on the release goal is sufficient and enables the product owner to have the latest information when he or she commits to the scope during sprint planning.

The product backlog and release plan are some of the most important communication channels between the product owner and the development team. In [Chapter 7](#), you find out how to complete a product backlog. How to create a release plan is described next.

The release plan contains a release schedule for a specific set of features. The product owner creates a release plan at the start of each release. To create a release plan, follow these steps:

- 1. Establish the release goal.**

The release goal is an overall business goal for the product features in your release. The product owner and development team collaborate to create a release goal based on business priorities and the development team's development speed and capabilities.

- 2. Identify a target release date.**

Some scrum teams determine release dates based on the completion of functionality; others may have hard dates, such as March 31 or September 1.

- 3. Review the product backlog and the product roadmap to determine the highest-priority user stories that support your release goal (the minimum marketable features).**

These user stories will make up your first release.



TIP We like to achieve releases with about 80 percent of the user stories, using the final 20 percent to add robust features that will meet the release goal while adding to the product's "wow" factor.

- 4. Refine the user stories in your release goal.**

During release planning, dependencies, gaps, or new details are often identified that affect estimates and prioritization. This is the time to make sure the portion of the product backlog supporting your release is sized appropriately. We like to make sure that requirements supporting the current release goal are sized no larger than 34. The development team helps the product owner by updating estimates for any added or revised user stories, and commits to the release goal and scope with the product owner.

5. **Estimate the number of sprints needed, based on the scrum team's velocity.**



TECHNICAL STUFF Scrum teams use velocity to plan how much work they can take on in a release and sprint. *Velocity* is the sum of all user story points completed within a sprint. So, if a scrum team completed six user stories during its first sprint with sizes 8, 5, 5, 3, 2, 1, their velocity for the first sprint is 24. The scrum team would plan its second sprint keeping in mind that it completed 24 story points during the first sprint.

After multiple sprints, scrum teams can use their running average velocity as an input to determine how much work they can take on in a sprint, as well as to extrapolate their release schedule by dividing the total number of story points in the release by their average velocity. You learn more about velocity in [Chapter 13](#).

6. **Identify work necessary to release that can't be completed within a sprint. Plan a release sprint, if necessary, and determine how long it should be.**



TIP Some project teams add a *release sprint* to some releases to conduct activities that are unrelated to product development but necessary to release the product to customers. If you need a release sprint, be sure to factor that into the date you choose. You can find more about release sprints in [Chapter 11](#).

Some tasks, such as security testing or load testing a software project,

can't be completed within a sprint, because the security or load testing environments take time to set up and request. Although release sprints allow scrum teams to plan for these types of activities, doing so is an anti-pattern, or the opposite of being agile. Your goal should be to complete all work required for functionality to be shippable at the end of each sprint.

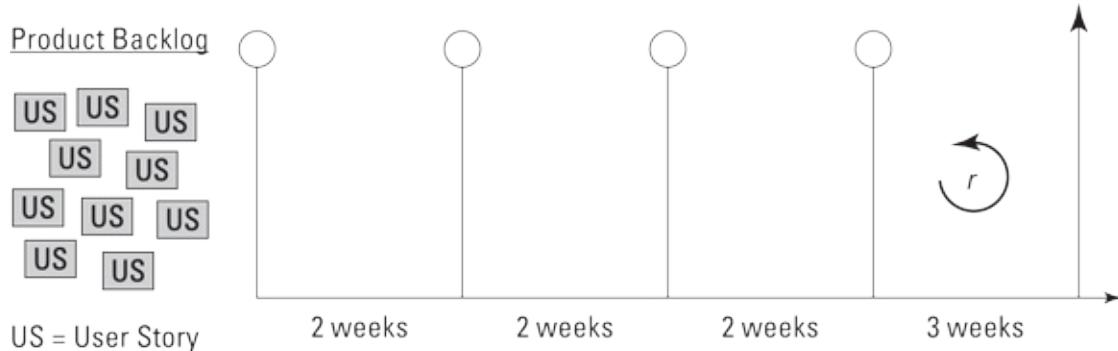


**TIP** Not all agile projects use release planning. Some scrum teams release functionality for customer use with every sprint, or even every day. The development team, product, organization, customers, stakeholders, and the project's technological complexity can all help determine your approach to product releases.

The planned releases now go from a tentative plan to a more concrete goal. [Figure 8-6](#) represents a typical release plan.

**Release Goal:** Enable customers to access, view, and transact against their active accounts

**Release Date:** March 31, 2021



[FIGURE 8-6:](#) Sample release plan.



**TIP** Bear in mind the pen-pencil rule: You can commit to (write in pen) the plan for the first release, but anything beyond the first release is tentative (written in pencil). In other words, use just-in-time planning (see [Chapter 7](#)) for each release. After all, things change, so why bother getting microscopic too early?

# Sprint Planning

In agile projects, a *sprint* is a consistent iteration of time in which the development team creates a specific group of product capabilities from start to finish. At the end of each sprint, the functionality that the development team has created should be working, ready to demonstrate, and potentially shippable to the customer.

Sprints should be the same length within a project. Keeping the sprint lengths consistent helps the development team measure its performance and plan better at each new sprint.

Sprints generally last one to four weeks. Four weeks is the longest amount of time any sprint should last; longer iterations make changes riskier, defeating the purpose of being agile. We rarely see sprints lasting longer than two weeks, and more often see sprints lasting a week. One-week sprints are a natural cycle with the Monday-to-Friday business week that structurally prevents weekend work. Some scrum teams work in one-day sprints where priorities change on a daily basis. Market and customer needs are changing more and more quickly, and the amount of time you can afford between opportunities to gather customer feedback only gets shorter. Our rule of thumb is that your sprint shouldn't be longer than your stakeholders can consistently go without changes in priority regarding what the scrum team should be working on in the sprint.

Each sprint includes the following:

- » Sprint planning at the beginning of the sprint
- » Daily scrum meetings
- » Development time — the bulk of the sprint
- » A sprint review and a sprint retrospective at the end of the sprint

Discover more about daily scrums, sprint development, the sprint review, and the sprint retrospective in [Chapters 9](#) and [10](#). In this chapter, you find out how to plan sprints.

Sprint planning is stage 4 in the Roadmap to Value, as you can see in [Figure 8-7](#). The entire scrum team — the Product owner, the scrum master, and the

development team — works together to plan sprints.

## Stage 4: SPRINT PLANNING

- Description:** Establishment of specific iteration goals and tasks
- Owner:** Product owner and development team
- Frequency:** At the start of each sprint

**FIGURE 8-7:** Sprint planning as part of the Roadmap to Value.

## *The sprint backlog*

The *sprint backlog* is a list of user stories associated with the current sprint and related tasks. When planning your sprint, you do the following:

- » Establish goals for your sprint.
  - » Choose the user stories that support those goals.
  - » Break user stories into specific development tasks.
  - » Create a *sprint backlog*. The sprint backlog consists of the following:
    - The list of user stories within the sprint in order of priority.
    - The relative effort estimate for each user story.
    - The tasks necessary to develop each user story.
    - The effort, in hours, to complete each task.

At the task level, you estimate the number of hours each task will take to complete, instead of using story points. Because your sprint has a specific length, and thus a set number of available working hours, you can use the time each task takes to determine whether the tasks will fit into your sprint.

Each task should take one day or less for the development team to complete.



**TIP** Some mature development teams may not need to estimate their tasks as they get more consistent at breaking down their user stories into executable tasks. Estimating tasks is helpful for newer

development teams to ensure that they understand their capacity and plan each sprint appropriately.

- A *burndown chart*, which shows the status of the work the development team has completed.



TECHNICAL STUFF

Tasks in agile projects should take a day or less to complete for two reasons. The first reason involves basic psychology: People are motivated to get to the finish line. If you have a task that you know you can complete quickly, you are more likely to finish it on time, just to check it off your to-do list. The second reason is that one-day tasks provide good red flags that a project might be veering off course. If a development team member reports that he or she is working on the same task for more than one or two days, that team member probably has a roadblock. The scrum master should take the opportunity to investigate what might be keeping the team member from finishing work. (For more on managing roadblocks, see [Chapter 9](#).) The development team collaborates to create and maintain the sprint backlog, and only the development team can modify the sprint backlog. The sprint backlog should reflect an up-to-the-day snapshot of the sprint's progress. [Figure 8-8](#) shows a sample sprint backlog at the end of the sprint planning meeting. You can use this example, find other samples, or even use a whiteboard.

MyXYZ Mobile Banking App – Sprint 1												
Sprint Dates: February 4 – February 8												
Sprint Goal												
Demonstrate the ability for personal banking users to view account balances and pending translations.												
<p>Burndown: Hours &amp; Points Remaining</p> <p>The chart shows a downward-sloping line starting at approximately 140 hours/points on Day 1 and ending near zero by Day 5. A legend indicates: Story Points (dashed line), Actual (solid line), Plan (dotted line).</p>			Available working hours in the sprint			Days in sprint : 4.5			Total			
Developers		M 4 (Start w/ sprint planning)	Tu 5	W 6	Th 7	F 8 (End w/ sprint review + retrospective)						
Suraj		4	5	6	6	4						
Nancy		4	5	6	6	4						
Kavita		4	6	6	6	4						
Liam		4	5	6	6	4						
Paul		4	5	6	6	4						
							Total sprint hours:	130				
							Total per day:	29				
Feature Burndown – Based on estimated hours remaining												
ID	Task	Story Points	Responsible	M 4 (Start w/ sprint planning)	Tu 5	W 6	Th 7	F 8 (End w/ sprint review + retrospective)	Done (Y)	Accepted (Y/N)		
125	<a href="#">View account balance</a>	8	Developer name	8	8	8	8	8	Y			
	Write automated unit test and develop API		Suraj	6	6	6	6	6				
	Implement UI		Nancy	3	3	3	3	3				
	Write automated functional test		Kavita	3	3	3	3	3				
	Write automated integration test		Paul	4	4	4	4	4				
	Write automated regression test		Liam	2	2	2	2	2				
	Conduct peer review			1	1	1	1	1				
	Update wiki			1	1	1	1	1				
	Promote to QA environment			1	1	1	1	1				
0059	<a href="#">View pending transactions</a>	5	Developer name	5	5	5	5	5				
	Write automated unit test			5	5	5	5	5				
	Implement UI			2	2	2	2	2				
	Write automated functional test			4	4	4	4	4				
	Write automated integration test			5	5	5	5	5				
	Write automated regression test			2	2	2	2	2				
	Conduct peer review			3	3	3	3	3				
	Update wiki			3	3	3	3	3				
	Promote to QA environment			3	3	3	3	3				

**FIGURE 8-8:** Sprint backlog example.

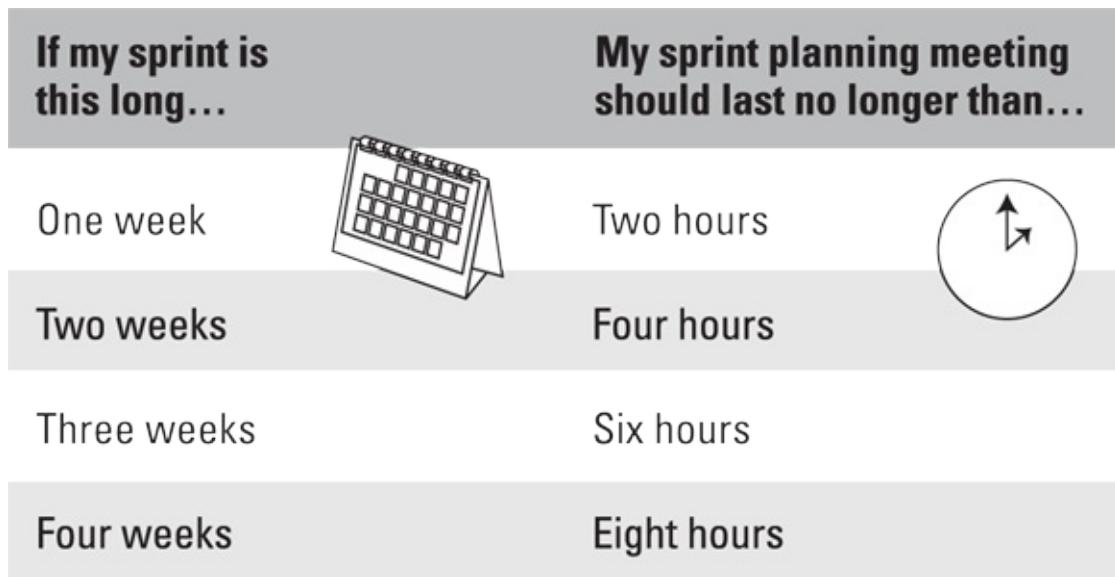
## The sprint planning meeting

On the first day of each sprint, often a Monday morning, the scrum team holds the sprint planning meeting.



**TIP** For a successful sprint planning meeting, make sure everyone involved in the session (the product owner, the development team, the scrum master, and anyone else the scrum team requests) is dedicated to the effort for the entire meeting.

Base the length of your sprint planning meeting on the length of your sprints: Meet for no more than two hours for every week of your sprints. This timebox is one of the rules of scrum and helps ensure that the meeting stays focused and on track. [Figure 8-9](#) illustrates this and is a good quick reference for your sprint planning meeting lengths.



**FIGURE 8-9:** Ratio of sprint planning meeting to sprint length.



TECHNICAL STUFF

On agile projects, the practice of limiting the time of your meetings is sometimes called *timeboxing*. Keeping your meetings timeboxed ensures that the development team has the time it needs to create the product.

You'll split your sprint planning meetings into two parts: one to set a sprint goal (the “why”) and choose user stories for the sprint (the “what”), and another to break down your user stories into individual tasks (the “how” and “how much”). The details on each part are discussed next.

### **Part 1: Setting goals and choosing user stories**

In the first part of your sprint planning meeting, the product owner and development team, with support from the scrum master, do the following:

1. Discuss and set a sprint goal.
2. Review the user stories from the product backlog that support the sprint goal and revisit their relative estimates.
3. If needed, create user stories to fill gaps to achieve the sprint goal.
4. Determine what the team can commit to in the current sprint.

At the beginning of your sprint planning meeting, the product owner should

propose a sprint goal and then together with the development team discuss and agree on the sprint goal. The sprint goal should be an overall description of the working customer functionality that the team will demonstrate and possibly release at the end of the sprint. The goal is supported by the highest-priority user stories in the product backlog. A sample sprint goal for the mobile banking application (refer to [Chapter 7](#)) might be as follows:

Demonstrate the ability of a mobile banking customer to log in and view account balances and pending and prior transactions.

Using the sprint goal, you determine the user stories that belong in the sprint. You also take another look at the estimates for those stories and make changes to the estimates if necessary. For the mobile banking application sample, the group of user stories for the sprint might include the following:

- » Log in and access my accounts.
- » View account balances.
- » View pending transactions.
- » View prior transactions.

All these would be high-priority user stories in the product backlog that support the sprint goal.

The second part of reviewing user stories is confirming that the effort estimates for each user story have been reviewed and adjusted if needed, and reflect the development team's current knowledge of the user story. Adjust the estimate if necessary. With the product owner in the meeting, resolve any outstanding questions. At the beginning of the sprint, the scrum team has the most up-to-date knowledge about the system and the customer's needs up to this point in the project, so make sure the development team and product owner have one more chance to clarify and size the user stories going into the sprint.

Finally, after you know which user stories support the sprint goal, the development team should agree and confirm that it can complete the goal planned for the sprint. If any of the user stories you discussed earlier don't fit in the current sprint, remove them from the sprint and add them back into the product backlog.



**WARNING** Always plan and work one sprint at a time. An easy trap to fall into is to place user stories into specific future sprints. For example, when you're still planning sprint 1, don't decide that user story X should go into sprint 2 or 3. Instead, keep the ordered list of user stories up to date in the product backlog and focus on always developing the next highest-priority stories. Commit to planning only for the current sprint.

After you have a sprint goal, user stories for the sprint, and a commitment to the goal, move on to the second part of sprint planning.



**TIP** Because a sprint planning meeting for sprints longer than one week might last a few hours, you might want to take a break between the two parts of the meeting.

## **Part 2: Breaking down user stories into tasks for the sprint backlog**

In the second part of the sprint planning meeting, the scrum team does the following:

1. The development team creates the sprint backlog tasks associated with each user story. Make sure that tasks encompass each part of the definition of done: developed, integrated, tested, and documented.
2. The development team double-checks that it can complete the tasks in the time available in the sprint.
3. Each development team member should choose his or her first task to accomplish before leaving the meeting.



**TIP** Development team members should each work on only one task on one user story at a time to enable *swarming* — the practice of the entire development team working on one user story until completion. Swarming can be an efficient way to complete work in a short amount of

time. In this way, scrum teams avoid getting to the end of the sprint with all user stories started but few finished.

At the beginning of part two of the meeting, break the user stories into individual tasks and allocate a number of hours to each task. The development team's target should be completing a task in a day or less. For example, a user story for the XYZ Bank mobile application might be as follows:

Log in and access my accounts.

The team decomposes this user story into tasks, such as the following:

- » Write the unit test.
- » Create an authentication screen for a username and password, with a Submit button.
- » Create an error screen for the user to reenter credentials.
- » Create a screen (once logged in) displaying a list of accounts.
- » Using authentication code from the online banking application, rewrite code for an iPhone/iPad application.
- » Create calls to the database to verify the username and password.
- » Refactor code for mobile devices.
- » Write the integration test.
- » Update the wiki documentation.

After you know the number of hours that each task will take, do a final check to make sure that the number of hours available to the development team reasonably matches the total of the tasks' estimates. If the tasks exceed the hours available, one or more user stories will have to come out of the sprint. Discuss with the product owner what tasks or user stories are the best to remove.

If extra time is available within the sprint, the development team might be able to include another user story. Just be careful about over-committing at the beginning of a sprint, especially in the project's first few sprints.

After you know which tasks will be part of the sprint, choose what you will

work on first. Each development team member should select his or her initial task to accomplish for the sprint. Team members should focus on one task at a time.



**TIP** As the development team members think about what they can complete in a sprint, use the following guidelines to ensure that they don't take on more work than they can handle while they're learning new roles and techniques:

- » **Sprint 1:** 25 percent of what the development team thinks it can accomplish. Include overhead for learning the new process and starting a new project.
- » **Sprint 2:** 50 percent of what the development team thinks it can accomplish.
- » **Sprint 3:** 75 percent of what the development team thinks it can accomplish.
- » **Sprint 4 and forward:** 100 percent. The development team will have developed a rhythm and velocity, gained insight into agile principles and the project, and will be working at close to full pace.

The scrum team should constantly evaluate the sprint backlog against the development team's progress on the tasks. At the end of the sprint, the scrum team can also assess estimation skills and capacity for work during the sprint retrospective (see [Chapter 10](#)). This evaluation is especially important for the first sprint.



**TIP** For the sprint, how many total working hours are available? In a 40-hour week, you could wisely assume, for a two-week sprint, that nine working days are available to develop user stories. If you assume each full-time team member has 35 hours per week (7 productive hours per day) to focus on the project, the number of working hours available is

$$\text{Number of team members} \times 7 \text{ hours} \times 9 \text{ days}$$

Why nine days? Half of day one is taken up with planning, and half of day ten is taken up with the sprint review (when the stakeholders review the completed work) and the sprint retrospective (when the scrum team identifies improvements for future sprints). That leaves nine days of development.

After the sprint planning is finished, the development team can immediately start working on the tasks to create the product!

The scrum master should make sure the product roadmap, product backlog, and sprint backlog are in a prominent place and accessible to everyone. This allows managers and other interested parties to view the artifacts and get the status of progress without interrupting the development team.

# Chapter 9

## Working throughout the Day

---

### IN THIS CHAPTER

- » Planning each day
- » Tracking daily progress
- » Developing and testing every day
- » Ending the day

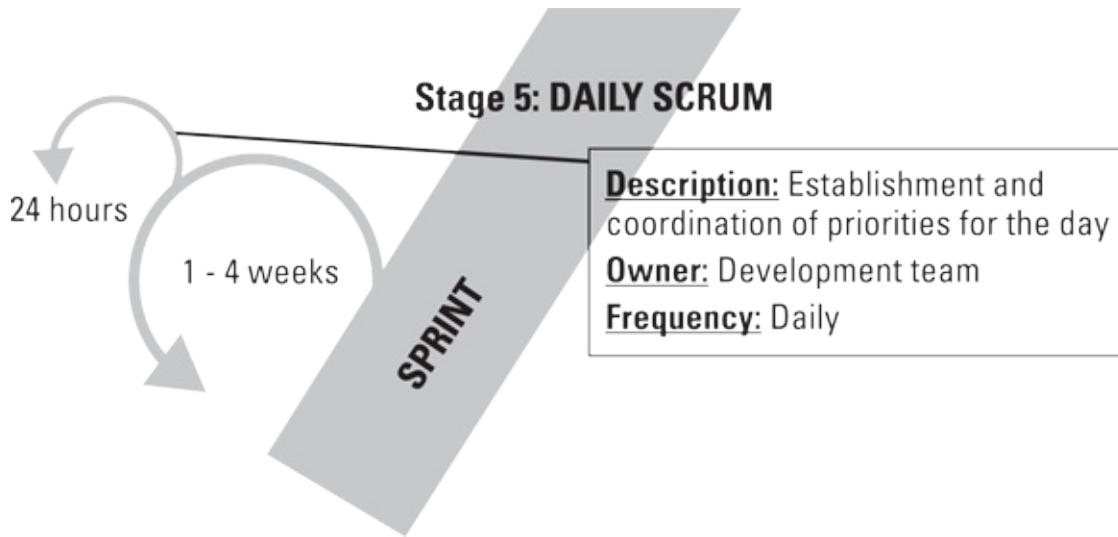
It's Tuesday, 9 a.m. Yesterday, you completed sprint planning, and the development team started work. For the rest of the sprint, you'll be working *cyclically*, where each day follows the same pattern.

In this chapter, you find out how to use agile principles daily throughout each sprint. You see the work that you will do every day as part of a scrum team: planning and coordinating your day, tracking progress, creating and verifying usable functionality, and identifying and dealing with impediments to your work. You see how the different scrum team members work together each day during the sprint to help create the product.

### ***Planning Your Day: The Daily Scrum***

On agile projects, you make plans throughout the entire project — and on a daily basis. Agile development teams start each workday with a *daily scrum* meeting to note completed items, to identify impediments, or roadblocks, requiring scrum master involvement, and to synchronize and plan what each team member will do during the day to achieve the sprint goal.

The daily scrum is Stage 5 on the Roadmap to Value. You can see how the sprint and the daily scrum fit into an agile project in [Figure 9-1](#). Note how they both repeat.



**FIGURE 9-1:** The sprint and the daily scrum in the Roadmap to Value.

In the daily scrum meeting, each development team member makes the following three statements, which enable team coordination:

- » **Yesterday, I completed** (state items completed).
- » **Today, I'm going to take on** (state task).
- » **My impediments are** (state impediments, if any).



TECHNICAL STUFF Other names you might hear for the daily scrum meeting are the *daily huddle* or the *daily standup* meeting. Daily scrum, daily huddle, and daily standup all refer to the same thing.

We also have the scrum master address these three statements regarding the team's impediments:

- » **Yesterday, I resolved to** (state impediments completed).
- » **Today, I'm going to work on removing** (state impediment).
- » **The impediments I'm going to escalate are** (state impediments you need assistance with, if any).

One of the rules of scrum is that the daily scrum meeting lasts 15 minutes or less; longer meetings eat into the development team's day. The meeting is

also referred to as the daily standup because standing encourages shorter meetings. You can also use props to keep daily scrum meetings quick.



TIP We start meetings by tossing a squeaky burger-shaped dog toy — don't worry; it's clean — to a random development team member. Each person makes his or her three statements and then passes the squeaky toy to someone else. If people are long-winded, we change the prop to a 500-page ream of copy paper, which weighs about five pounds. Each person can talk for as long as he or she can hold the ream out to one side. Either meetings will quickly become shorter, or development team members will quickly build up their arm strength — in our experience, it's the former.

To keep daily scrums brief and effective, the scrum team can follow several guidelines:

- » **Anyone may attend a daily scrum, but only the development team, the scrum master, and the product owner may talk.** The daily scrum is the scrum team's opportunity to coordinate daily activities, not take on additional requirements or changes from stakeholders. Stakeholders can discuss questions with the scrum master or product owner afterward, but stakeholders should not approach the development team.
- » **The focus is on immediate priorities.** The scrum team should review only completed tasks, tasks to be done, and roadblocks.
- » **Daily scrum meetings are for coordination, not problem-solving.** The development team and the scrum master are responsible for removing roadblocks during the day.
- » **To keep meetings from drifting into problem-solving sessions, scrum teams can**
  - Create a list on a whiteboard to keep track of issues that need immediate attention, and then address those issues directly after the meeting with only those team members who need to be involved.
  - Hold a meeting, called an *after-party*, to solve problems after the daily scrum is finished. Some scrum teams schedule time for an

after-party every day; others meet only as needed.

- » **The daily scrum is for peer-to-peer coordination.** It is not used for an individual to report status to one person, such as the scrum master or product owner. Status is reported at the end of each day in the sprint backlog.
- » **Such a short meeting must start on time.** It is not unusual for the scrum team to have creative punishments for tardiness (such as doing pushups or adding penalty money into a team celebration fund or another inconvenience). Whatever method is used, the scrum team agrees on it together; the method is not dictated to them by someone outside the team, such as a manager.
- » **The scrum team may request that daily scrum attendees stand up — rather than sit down — during the meeting.** Standing up makes people eager to finish the meeting and get on with the day's work.



TIP When you have only 15 minutes to meet, every minute counts. Scrum teams should not be afraid to make being late to the daily scrum appropriately unpleasant. If members of the team love to sing, for example, performing a karaoke song probably won't have much of an effect. We've helped cure perpetual tardiness problems overnight by suggesting that the scrum team change its punishment from a \$1 to a \$20 contribution to the team celebration fund.

Daily scrum meetings are effective for keeping the development team focused on the right tasks for any given day. Because the development team members are accountable for their work in front of their peers, they are less likely to stray from their daily commitments. Daily scrum meetings also help ensure that the scrum master and development team can deal with roadblocks immediately. These meetings are so useful that even organizations that are not using any other agile techniques sometimes adopt daily scrums.



TIP We like to hold daily scrum meetings one hour after the development

team's normal start time to allow for traffic jams, emails, coffee, and other rituals when starting the day. Having a later scrum meeting also allows the development team time to review defect reports from automated testing tools that were run the night before.

The daily scrum is for discussing progress and planning each upcoming day. As you see next, you also track progress — not just discuss it — every day.

## Tracking Progress

You also need to track the progress of your sprint daily. This section discusses ways to keep track of the tasks in your sprint.

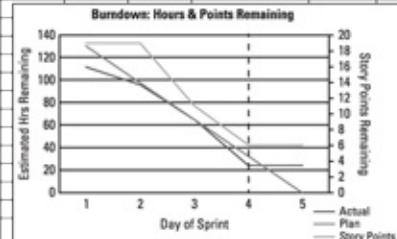
Two tools for tracking progress are the sprint backlog and a task board. Both the sprint backlog and the task board enable the scrum team to show the sprint's progress to anyone at any given time.



**REMEMBER** The Agile Manifesto values individuals and interactions over processes and tools. Make sure your tools support, rather than hinder, your scrum team. Modify or even replace tools if you have to. Read more about the Agile Manifesto in [Chapter 2](#).

### *The sprint backlog*

During sprint planning, you concentrate on adding user stories and tasks to the sprint backlog. During the sprint itself, you update the sprint backlog daily, tracking progress of the development team's tasks for each working day. [Figure 9-2](#) shows the sprint backlog for this book's sample application, the XYZ Bank's mobile banking application, as it would appear after day 4 of the first sprint. ([Chapter 8](#) discusses details of the sprint backlog.)

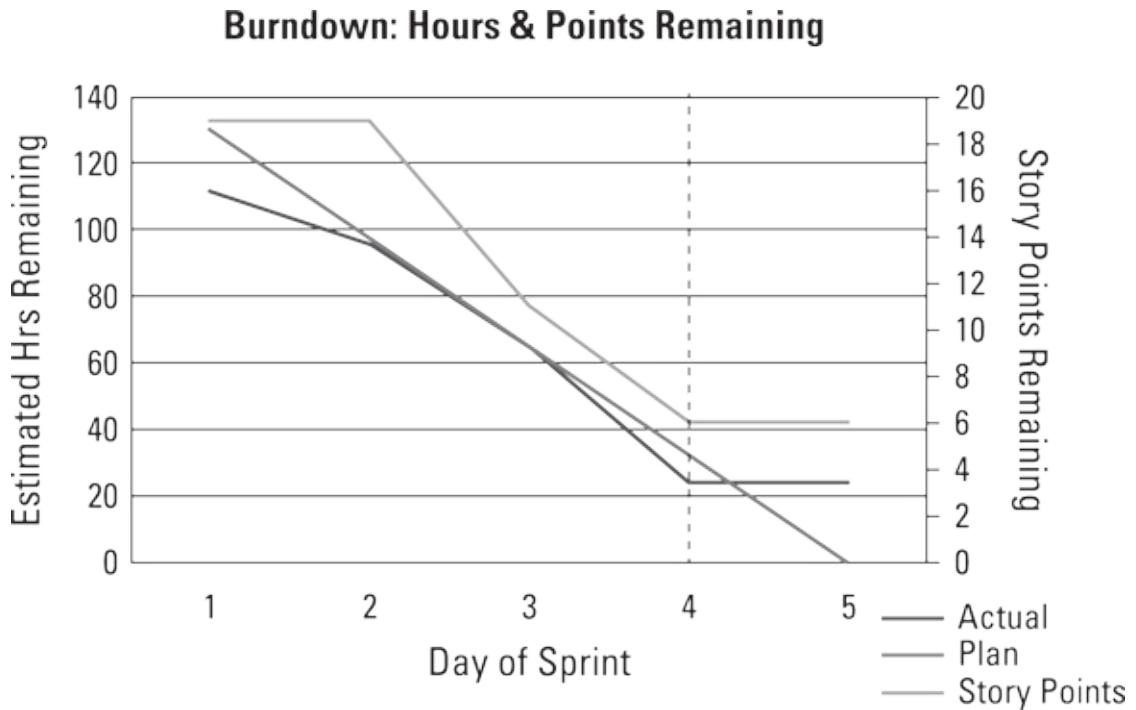
MyXYZ Mobile Banking App – Sprint 1										
Sprint Dates: February 4 – February 8										
Sprint Goal										
Demonstrate the ability for personal banking users to view account balances and pending translations.										
 <p>Burndown: Hours &amp; Points Remaining</p> <p>The burndown chart shows the progress of a sprint. The Y-axis represents 'Estimated Hrs Remaining' from 0 to 140, and the X-axis represents 'Day of Sprint' from 1 to 5. Three lines are plotted: 'Actual' (red), 'Plan' (blue), and 'Story Points' (green). The 'Actual' line starts at approximately 130 and ends at 0 by day 5. The 'Plan' line starts at 140 and ends at 0 by day 5. The 'Story Points' line starts at 20 and ends at 0 by day 5. All three lines follow a similar downward trajectory, indicating a consistent burn rate.</p>										
Feature Burndown – Based on estimated hours remaining										
ID	Task	Story Points	Responsible	M 4 (Start w/ sprint planning)	Tu 5	W 6	Th 7	F 8 (End w/ sprint review + retrospective)	Done (Y)	Accepted (Y/N)
125	<a href="#">View account balance</a>	8	Developer name	8	8	0	0	0	Y	Y
	Write automated unit test and develop API		Suraj	6	0	0	0	0		
	Implement UI		Nancy	3	0	0	0	0		
	Write automated functional test		Kavita	3	0	0	0	0		
	Write automated integration test		Paul	4	0	0	0	0		
	Write automated regression test		Suraj	2	1	0	0	0		
	Conduct peer review		Nancy	1	1	0	0	0		
	Update wiki		Paul	1	1	0	0	0		
	Promote to QA environment		Nancy	1	1	0	0	0		
0059	<a href="#">View pending transactions</a>	5	Developer name	5	5	0	0	0	Y	Y
	Write automated unit test		Kavita	5	5	0	0	0		
	Implement UI		Paul	2	2	0	0	0		
	Write automated functional test		Suraj	4	4	0	0	0		
	Write automated integration test		Nancy	5	5	0	0	0		
	Write automated regression test		Nancy	2	2	0	0	0		
	Conduct peer review		Suraj	3	3	0	0	0		
	Update wiki		Paul	3	3	0	0	0		
	Promote to QA environment		Kavita	3	3	1	0	0		

**FIGURE 9-2:** Sample sprint backlog.

Make the sprint backlog available to the entire project team every day. That way, anyone who needs to know the sprint status can find it instantly.

Near the top left of [Figure 9-2](#), note the *sprint burndown chart*, which shows the progress that the development team is making. You can see that the development team members have completed tasks close to the even burn rate of their available hours, and the product owner has accepted several user stories as complete.

You can include burndown charts on your sprint backlog and on your product backlog. (This chapter concentrates on the sprint backlog.) [Figure 9-3](#) shows the burndown chart in detail.



**FIGURE 9-3:** A burndown chart.

A burndown chart is a powerful tool for visualizing progress and the work remaining. The chart shows the following:

- » The outstanding work (in hours) on the first vertical axis
- » Time, in days along the horizontal axis

Some sprint burndown charts, like the one in [Figure 9-3](#), also show the outstanding story points on a second vertical axis that is plotted against the same horizontal time axis as hours of work remaining.

A burndown chart enables anyone, at a glance, to see the status of the sprint. Progress is clear. By comparing the realistic number of hours available to the work remaining, you can find out daily whether the effort is going as planned, is in better shape than expected, or is in trouble. This information helps you determine whether the development team is likely to accomplish the targeted number of user stories and helps you make informed decisions early in the sprint.



ON THE  
WEB

You can create a sprint backlog using a spreadsheet and charting program such as Microsoft Excel. You can also download our sprint backlog template, which includes a burndown chart, from the book's website at [www.dummies.com/go/agileprojectmanagementfd2e](http://www.dummies.com/go/agileprojectmanagementfd2e).

**Figure 9-4** shows samples of burndown charts for sprints in different situations. Looking at these charts, you can tell how the work is progressing:

- » **1. Expected:** This chart shows a normal sprint pattern. The remaining work hours rise and fall as the development team completes tasks, ferrets out details, and identifies tactical work it may not have initially considered. Although work occasionally increases, it is manageable, and the team mobilizes to complete all user stories by the end of the sprint.
- » **2. More complicated:** In this sprint, the work increased beyond the point at which the development team felt it could accomplish everything. The team identified this issue early, worked with the product owner to remove some user stories, and still achieved the sprint goal. The key to scope changes within a sprint is that they are always initiated by the development team — no one else.
- » **3. Less complicated:** In this sprint, the development team completed some critical user stories faster than anticipated and worked with the product owner to identify additional user stories it could add to the sprint.
- » **4. Not participating:** A straight line in a burndown means that the team didn't update the burndown or made zero progress that day. Either case is a red flag for future problems.



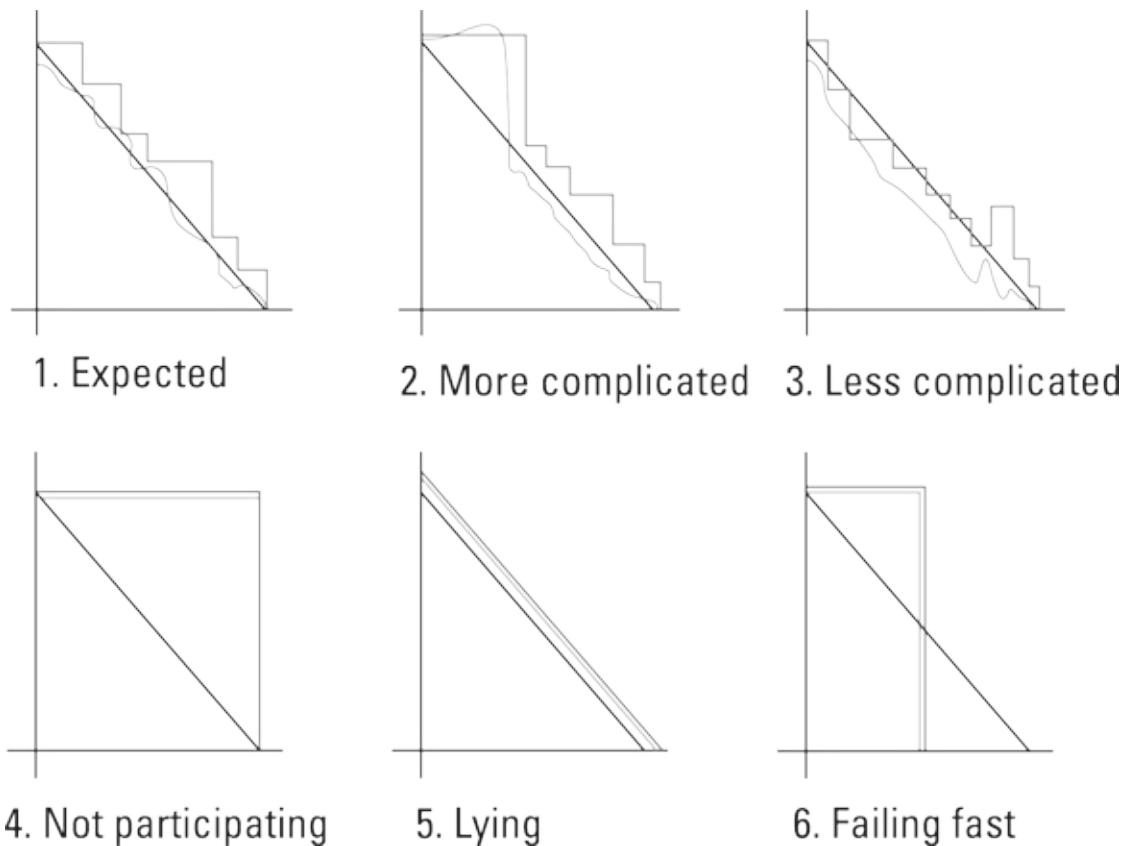
WARNING

Just like on a heartbeat graph, a horizontal straight line on a sprint burndown chart is never a good thing.

- » **5. Lying (or conforming):** This burndown pattern is common for a new agile development team that might be accustomed to reporting the hours that management expects, instead of the time the work really takes, and consequently tends to adjust the team's work estimates to the exact

number of remaining hours. This pattern often reflects a fear-based environment, where the managers lead by intimidation.

- » **6. Failing fast:** One of the strongest benefits of this simple visualization of progress is the immediate proof of progress or lack thereof. This pattern shows an example of a team that wasn't participating or progressing. Halfway through the sprint, the product owner decided to cut losses by killing the sprint and starting a new sprint with a new sprint goal. Only product owners can end a sprint early.



**FIGURE 9-4:** Profiles of burndown charts.

The sprint backlog helps you track progress throughout each sprint. You can also refer to earlier sprint backlogs to compare progress from sprint to sprint. You make changes to your process in each sprint (read more about the concept of inspect and adapt in [Chapter 10](#)). Constantly inspect your work and adapt to make it better. Hold on to those old sprint backlogs.

Another way to keep track of your sprint is by using a task board. Read on to find out how to create and use one.

## The task board

Although the sprint backlog is a great way to track and show project progress, it's probably in an electronic format, so it might not be immediately accessible to anyone who wants to see it. Some scrum development teams use a task board along with their sprint backlog. A *task board* provides a quick, easy view of the items in the sprint that the development team is working on and has completed.

We like task boards because you can't deny the status they show. Like the product roadmap, the task board can be made up of sticky notes on a whiteboard. The task board will have at least the following four columns, from left to right:

- » **To Do:** The user stories and tasks that remain to be accomplished are in the far left column.
- » **In Progress:** User stories and tasks that the development team is currently working on are in the In Progress column. Only one user story should be in this column. Having more user stories in progress is an alert that development team members are not working cross-functionally and, instead, are hoarding desired tasks. You risk having multiple user stories partially done instead of more user stories completely done by the end of the sprint.
- » **Accept:** After the development team completes a user story, it moves it to the Accept column. User stories in the Accept column are ready for the product owner to review and either provide feedback or accept.
- » **Done:** When the product owner has reviewed a user story and verifies that the user story is complete, the product owner can move that user story to the Done column.



TIP Limit your work in progress! Only select one task at a time. Leave other tasks available in the To Do column. Ideally, a development team will work on only one user story at a time and swarm on the tasks of that user story to complete it quickly.

Because the task board is tactile — people physically move a user story card

through its completion — it can engage the development team more than an electronic document ever could. The task board encourages thought and action just by existing in the scrum team's work area, where everyone can see the board.



**TIP** Allowing only the product owner to move user stories to the Done column prevents misunderstandings about user story status.

[Figure 9-5](#) shows a typical task board. As you can see, the task board is a strong visual representation of the work in progress.

**RELEASE GOAL:**

**SPRINT GOAL:**

**RELEASE DATE:**

**SPRINT REVIEW:**

= User Story

= Task

TO DO	IN PROGRESS	ACCEPT	DONE
			US Task  Task  Task  Task  Task  Task Task  Task  Task  Task  Task  Task
		US	Task  Task  Task  Task  Task  Task Task  Task  Task  Task  Task  Task
Task  Task Task  Task Task  Task Task  Task	US  Task  Task  Task Task  Task		
US  Task  Task Task  Task Task  Task Task  Task			

[FIGURE 9-5:](#) Sample task board.



TECHNICAL STUFF

The task board is a lot like a kanban board. *Kanban* is a Japanese term that means *visual signal*. (For more on kanban boards, see [Chapter 4](#).) Toyota created these boards as part of its lean manufacturing process.

In [Figure 9-5](#), the task board shows four user stories, each separated by a horizontal line called *swim lanes*. The first user story is done. All tasks are completed, and the product owner has accepted the work done. For the second user story, the development work is completed but is waiting for acceptance by the product owner. The third user story is in progress, and the fourth user story has not yet been started. At a glance, the status of each user story is clear not only to the scrum team, making tactical coordination faster and easier, but also to interested stakeholders.

Day-to-day work on an agile project involves more than just planning and tracking progress. In the next section, you see what most of your day's work will include, whether you're a member of the development team, a product owner, or a scrum master.



TIP

Some development teams report status only with a task board and ask the scrum master to convert the status into the sprint backlog. This process helps the scrum master see trends and potential issues.

## ***Agile Roles in the Sprint***

Each member of a scrum team has specific daily roles and responsibilities during the sprint. The day's focus for the development team is producing shippable functionality. For the product owner, the focus is on preparing the product backlog for future sprints while supporting the development team's execution of the sprint backlog with real-time clarifications. The scrum master is the agile coach and maximizes the development team's productivity by removing roadblocks and protecting the development team from external distractions.

Following are descriptions of the tasks each member of the scrum team

performs during the sprint. If you're a member of the development team, you

- » Select the tasks of highest need and complete them as quickly as possible.
- » Request clarification from the product owner when you are unclear about a user story.
- » Collaborate with other development team members to design the approach to a specific user story, seek help when you need it, and provide help when another development team member needs it.
- » Conduct peer reviews on one another's work.
- » Take on tasks beyond your normal role as the sprint demands.
- » Fully develop functionality as agreed to in the definition of done (described in the next section, “[Creating Shippable Functionality](#)”).
- » Report daily on your progress completing tasks in the sprint backlog.
- » Alert the scrum master to any roadblocks you can't effectively remove on your own.
- » Achieve the sprint goal you committed to during sprint planning.

The product owner has the following tasks during the sprint:

- » Make investments required to keep development speed high.
- » Prioritize product functionality.
- » Represent the product stakeholders to the development team.
- » Report on cost and schedule status to project stakeholders.
- » Elaborate user stories with the development team so that the team clearly understands what it is creating.
- » Provide immediate clarification and decisions about requirements to keep the development team developing.
- » Remove business impediments brought to you by other members of the scrum team.
- » Review completed functionality for user stories and provide feedback to the development team.
- » Add new user stories to the product backlog as necessary and ensure that

new user stories support the product vision, the release goal, and the sprint goal.

- » Look forward to the next sprint and elaborate user stories in readiness for the next sprint planning meeting.



**REMEMBER** Nonverbal communication says a lot. Scrum masters can benefit from understanding body language to identify unspoken tensions in the scrum team.

If you're a scrum master, you do the following during the sprint:

- » Uphold agile values and practices by coaching the product owner, the development team, and the organization when necessary.
- » Shield the development team from external distractions.
- » Remove roadblocks, both tactically for immediate problems and strategically for potential long-term issues. In [Chapter 6](#), we compare the scrum master to an aeronautical engineer, continually removing and preventing organizational drag on the development team.
- » Facilitate consensus building in the scrum team.
- » Build relationships to foster close cooperation with people working with the scrum team.



**TIP** We often tell scrum masters, "Never lunch alone. Always be building relationships." You never know when you will need to call in a favor on a project.

As you can see, each scrum team member has a specific job in the sprint. In the next section, you see how the product owner and the development team work together to create the product.

## ***Creating Shippable Functionality***

The objective of the day-to-day work of a sprint is to create shippable functionality for the product in a form that can be delivered to a customer or user.

Within the context of a single sprint, a *product increment* or *shippable functionality* means that a work product has been developed, integrated, tested, and documented according to the project definition of done and is deemed ready to release. The development team may or may not release that product at the end of the sprint — release timing depends on the release plan. The project may require multiple sprints before the product contains the set of minimum marketable features necessary to justify a market release.



TIP It helps to think about shippable functionality in terms of user stories. A user story starts out as a written requirement on a card. As the development team creates functionality, each user story becomes an action a user can take. Shippable functionality equals completed user stories.

To create shippable functionality, the development team and the product owner are involved in three major activities:

- » Elaborating
- » Developing
- » Verifying

During the sprint, any or all of these activities can be happening at any given time. As you review them in detail, remember that they don't always occur in a linear way.

## ***Elaborating***

In an agile project, *elaboration* is the process of determining the details of a product feature. Whenever the development team tackles a new user story, elaboration ensures that any unanswered questions about a user story are answered so that the process of development can proceed.

The product owner works with the development team to elaborate user stories, but the development team should have the final say on design

decisions. The product owner should be available for consultation if the development team needs further clarification on requirements throughout the day.



**WARNING** Collaborative design is a major factor for successful projects.

Remember these agile principles: “The best architectures, requirements, and designs emerge from self-organizing teams,” and “Business people and developers must work together daily throughout a project.” Watch out for development team members who have a tendency to try to work alone on elaborating user stories. If a member of the development team separates himself or herself from the team, perhaps part of the scrum master’s job should be coaching that person on upholding agile values and practices.

## *Developing*

During product development, most of the activity, naturally, falls to the development team. The product owner continues to work with the development team on an as-needed basis to provide clarification and to approve developed functionality.



**TIP** The development team should have immediate access to the product owner. Ideally, the product owner sits with the development team when he or she is not interacting with customers and stakeholders.

The scrum master should focus on protecting the development team from outside disruptions and removing impediments that the development team encounters.

To sustain agile practices during development, be sure to implement the type of development practices from extreme programming we show you in [Chapter 4](#), including the following:

- » **Pair up development team members to complete tasks.** Doing so enhances the quality of the work and encourages the sharing of skills.

- » **Follow the development team's agreed-upon design standards.** If you can't follow them for whatever reason, revisit these standards and improve them.
- » **Start development by setting up automated tests.** You can find more about automated testing in the following section and in [Chapter 15](#).
- » **If new, nice-to-have features become apparent during development, add them to the product backlog.** Avoid coding new features that are outside the sprint goal.
- » **Integrate changes that were coded during the day, one set at a time.** Test for 100 percent correctness. Integrate changes at least once a day; some teams integrate many times a day.
- » **Undertake code reviews to ensure that the code follows development standards.** Identify areas that need revising. Add the revisions as tasks in the sprint backlog.
- » **Create technical documentation as you work.** Don't wait until the end of the sprint or, worse, the end of the sprint prior to a release.



TECHNICAL STUFF

*Continuous integration* is the term used in software development for integrating and comprehensively testing with every code build. Continuous integration helps identify problems before they become crises.

## Verifying

Verifying the work done in a sprint has three parts: automated testing, peer review, and product owner review.



TIP It is exponentially cheaper to prevent a defect than it is to rip it out of a deployed system.

## Automated testing

*Automated testing* means using a computer program to do the majority of

your code testing for you. With automated testing, the development team can quickly develop and test code, which is a big benefit for agile projects.

Often, agile project teams code during the day and let the tests run overnight. In the morning, the project team can review the defect report that the testing program generated, report on any problems during the daily scrum, and correct those issues immediately during the day.

Automated testing can include the following:

- » **Unit testing:** Testing source code in its smallest parts — the component level
- » **System testing:** Testing the code with the rest of the system
- » **Static testing:** Verifying that the product's code meets standards based on rules and best practices that the development team has agreed upon

## **Peer review**

*Peer review* simply means that development team members review one another's code. If Sam writes program A and Joan writes program B, Sam could review Joan's code, and vice versa. Objective peer review helps ensure code quality.

*Pair programming* is another form of peer review, but the review takes place during development. While one developer (the pilot) sits at the keyboard and writes the code, another developer (the navigator) is thinking strategically, looking ahead, and actively listening and responding to decisions made tactically by the pilot. Not only is the review happening in the moment — catching defects and making more informed decisions — but there are two developers, instead of only one, who are intimately familiar with the part of the system being developed.



TIP Pair programming is a great way to develop cross-functional individuals to reduce single points of failure.

The development team can conduct peer reviews during development. Collocation helps make this easy — you can turn to the person next to you and ask him or her to take a quick look at what you just completed. The

development team can also set aside time during the day specifically for reviewing code. Self-managing teams should decide what works best for them.

### **Product owner review**

When a user story has been developed and tested, the development team moves the stories to the Accept column on the task board. The product owner then reviews the functionality and verifies that it meets the goals of the user story, per the user story's acceptance criteria. The product owner verifies user stories throughout each day.

As discussed in [Chapter 8](#), the back side of each user story card has verification steps. These steps allow the product owner to review and confirm that the code works and supports the user story. [Figure 9-6](#) shows a sample user story card's verification steps.

When I do this:	This happens:
When I go to the accounts page :	I am able to see my active account balance.
When I select transfer funds :	I am able to select "Transfer to Account" and amount.
When I submit transfer requests :	I get an account confirmation funds were transferred.

**FIGURE 9-6:** User story verification.

Finally, the product owner should run through some checks to verify that the user story in question meets the definition of done. When a user story meets the definition of done, the product owner updates the task board by moving the user story from the Accept column to the Done column.

While the product owner and the development team are working together to create shippable functionality for the product, the scrum master helps the scrum team to identify and clear roadblocks that appear along the way.

### **Identifying roadblocks**

It's a major part of the scrum master's role to manage and help resolve roadblocks that the scrum team identifies. Roadblocks are anything that thwarts a team member from working to full capacity.

Although the daily scrum is a good place for the development team to identify roadblocks, the development team may come to the scrum master with issues anytime throughout the day.

Examples of roadblocks follow:

» **Local, tactical issues**, such as

- A manager trying to pull away a team member to work on a “priority” sales report.
- The development team needing additional hardware or software to facilitate progress.
- A development team member who doesn’t understand a user story and says the product owner isn’t available to help.

» **Organizational impediments**, such as

- An overall resistance to agile techniques, especially when the company established and maintained prior processes at significant cost.
- Managers who might not be in touch with the work on the ground. Technologies, development practices, and project management practices are always progressing.
- External departments that may not be familiar with scrum needs and the pace of development when using agile techniques.
- An organization that enforces policies that don’t make sense for agile project teams. Centralized tools, budget restrictions, and standardized processes that don’t align with agile processes can all cause issues for agile teams.



**REMEMBER** The most important trait a scrum master can have is organizational clout or influence. Organizational clout gives the scrum master the ability to have difficult conversations and make the small and large changes necessary for the scrum team to be successful. We provide examples of different types of clout in [Chapter 4](#).

Beyond the primary focus of creating shippable functionality, other things happen during the day on an agile project. Many of these tasks fall to the scrum master. [Table 9-1](#) shows potential roadblocks and the action that the scrum master can take to remove the impediments.

**TABLE 9-1 Common Roadblocks and Solutions**

Roadblock	Action
The development team needs simulation software for a range of mobile devices so that it can test the user interface and code.	Do some research to estimate the cost of the software, prepare a summary of that for the product owner, and have a discussion about funding. Process the purchase through procurement, and deliver the software to the development team.
Management wants to borrow a development team member to write a couple of reports. All your development team members are fully occupied.	Tell the requesting manager that the person is not available and probably will not be for the duration of the project. Recommend that the requester discuss the need with the product owner so he or she can prioritize it against the rest of the product backlog. As you're likely a problem solver, you may want to suggest alternative ways in which the manager could get what he or she needs.
A development team member can't move forward on a user story because he or she does not fully understand the story. The product owner is out of the office for the day on a personal emergency.	Work with the development team member to determine if any work can happen around this user story while waiting for an answer. Help locate another person who could answer the question. Failing that, ask the development team to review upcoming tasks (not related to this stopped one) and move things around to keep productivity up.
A user story has grown in complexity and now appears to be too large for the sprint length.	Have the development team work with the product owner to break the user story down so that some demonstrable value can be completed in the current sprint and the rest can be put back into the product backlog. The goal is to ensure that the sprint ends with completed user stories, even smaller ones, rather than incomplete user stories.

So far in this chapter, you have seen how the scrum team starts its day and works throughout the day. The scrum team wraps up each day with a few tasks as well. The next section shows you how to end a day within a sprint.

## *The End of the Day*

At the end of each day, the development team reports on task progress by updating the sprint backlog with which tasks were completed and how much work, in hours, remains to be done on new tasks started. Depending on the software that the scrum team uses, the sprint backlog data may automatically update the sprint burndown chart as well.



**WARNING** Update the sprint backlog with the amount of work remaining — not the amount of time already spent — on open tasks. The important point is how much time is left, which informs the project team as to whether the scrum team is on track to meet its sprint goal. If possible, avoid spending time tracking how many hours were spent working on tasks, which is less necessary with self-correcting agile models.

The product owner should also update the task board, at least at the end of the day, and move any user stories that have passed review to the Done column.

The scrum master can review the sprint backlog or task board for any risks before the next day's daily scrum.

The scrum team follows this daily cycle until the end of the sprint, when it will be time to step back, inspect, and adapt at the sprint review and the sprint retrospective meetings.

# Chapter 10

## Showcasing Work, Inspecting, and Adapting

---

### IN THIS CHAPTER

- » Showcasing work and collecting feedback
- » Reviewing the sprint and improving processes

At the end of each sprint, the scrum team gets a chance to put the results of its hard work on display in the sprint review. The sprint review is where the product owner and the development team demonstrate the sprint's completed, potentially shippable functionality to the stakeholders. In the sprint retrospective, the scrum team (the product owner, development team, and scrum master) review how the sprint went and determine whether they need any adjustments for the next sprint.

Underpinning both of these events is the agile concept of inspect and adapt, which [Chapter 7](#) explains.

In this chapter, you find out how to conduct a sprint review and a sprint retrospective.

### *The Sprint Review*

The *sprint review* is a meeting to review and demonstrate the functionality created from the user stories that the development team completed during the sprint, and for the product owner to collect feedback and update the product backlog accordingly. The sprint review is open to anyone interested in reviewing the sprint's accomplishments. This means that all stakeholders get a chance to see product progress and accuracy, and provide feedback.

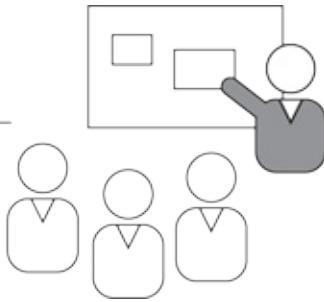
The sprint review is stage 6 in the Roadmap to Value. [Figure 10-1](#) shows how the sprint review fits into an agile project.

## Stage 6: SPRINT REVIEW

**Description:** Demonstration of working product

**Owner:** Product owner and development team

**Frequency:** At the end of each sprint



**FIGURE 10-1:** The sprint review in the Roadmap to Value.

The following sections show you what you need to do to prepare for a sprint review, how to run a sprint review meeting, and the importance of collecting feedback.

### *Preparing to demonstrate*

Preparation for the sprint review meeting should not take more than a few minutes at most. Even though the sprint review might sound formal, the essence of showcasing for agile teams is informality. The meeting needs to be prepared and organized, but it doesn't require a lot of flashy materials. Instead, the sprint review focuses on demonstrating what the development team has done.



**WARNING** If your sprint review is overly showy, ask yourself if you're covering up for not spending enough time developing. Get back to working on value — creating a working product. Pageantry is the enemy of agility.

The preparation for the sprint review meeting involves the product owner and the development team, facilitated by the scrum master as needed. The product owner needs to know which user stories the development team completed during the sprint. The development team needs to be ready to demonstrate completed, shippable functionality.

The time needed to prepare for a sprint review should not be more than 20 minutes — just enough time to make sure everyone knows who is doing what and when so the demonstration goes smoothly.



**REMEMBER** Work not delivered has no business value. Within the context of a single sprint, *shippable functionality* means that the development team has satisfied its definition of done for each requirement, and the product owner has verified that the work product meets all acceptance criteria and could be released to the market, or *shipped*, if the value and timing are right for the marketplace. The actual release may be at a later time, per the communicated release plan. Find out more about shippable functionality in [Chapter 9](#).

For the development team to demonstrate the code in the sprint review, it must be complete according to the definition of done. This means that the code is fully

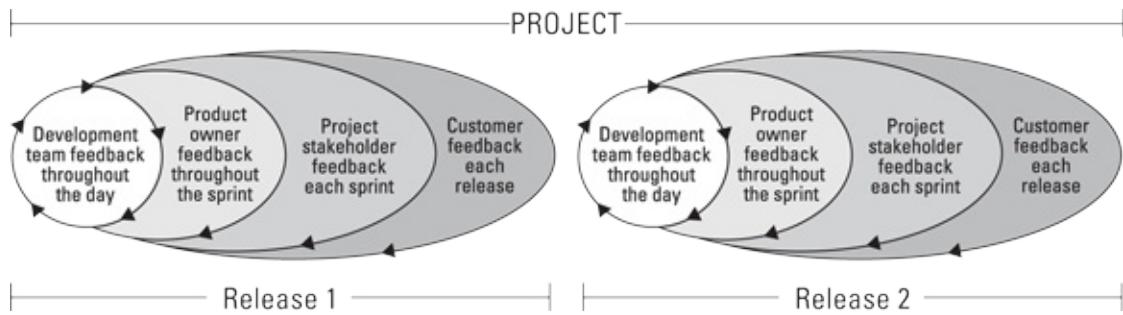
- » Developed
- » Tested
- » Integrated
- » Documented

As user stories are moved to a status of done throughout the sprint, the product owner and development team should check that the functionality meets these standards. This continuous validation throughout the sprint reduces end-of-sprint risks and helps the scrum team spend as little time as possible preparing for the sprint review.

Knowing the completed user stories and being ready to demonstrate those stories' functionality prepares you to confidently start the sprint review meeting.

## ***The sprint review meeting***

Sprint review meetings have two activities: demonstrate and showcase the scrum team's finished work, and allow stakeholders to provide feedback on that work. [Figure 10-2](#) shows the different loops of feedback a scrum team receives about a product.

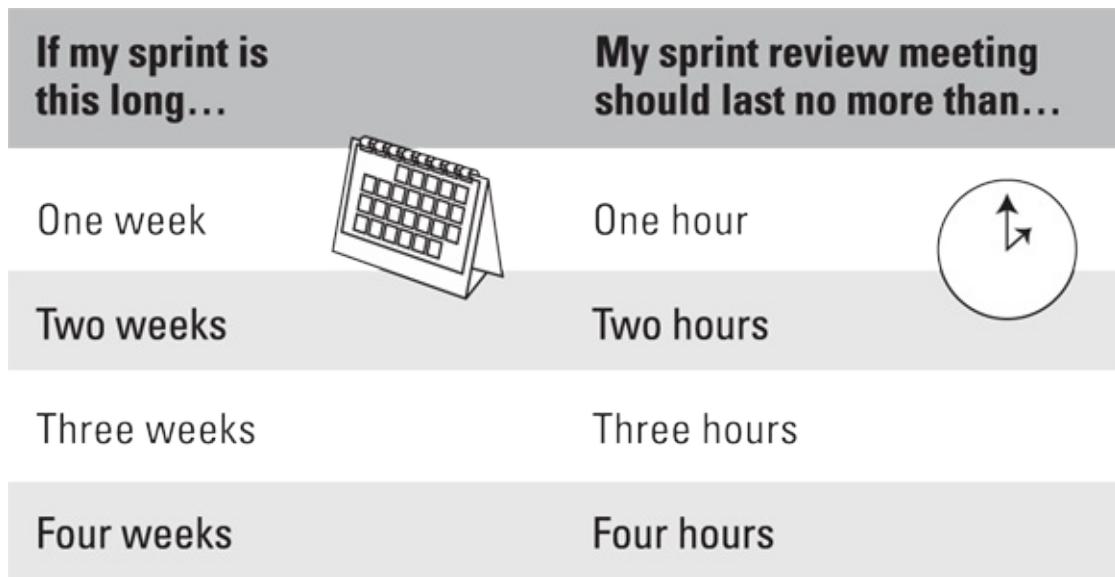


**FIGURE 10-2:** Agile project feedback loops.

This cycle of feedback repeats throughout the project as follows:

- » Each day, development team members work together in a collaborative environment that encourages feedback through peer reviews and informal communication.
- » Throughout each sprint, as soon as the development team completes each requirement, the product owner provides feedback by reviewing the working functionality for acceptance. The development team then immediately incorporates that feedback, if any, to satisfy the user story's acceptance criteria. When the user story is complete, the product owner gives final acceptance of the functionality created for the user story, according to the user story's acceptance criteria.
- » At the end of each sprint, project stakeholders provide feedback about completed functionality in the sprint review meeting.
- » With each release, customers who use the product provide feedback about new working functionality.

The sprint review usually takes place later in the day on the last day of the sprint, often a Friday. One of the rules of scrum is to spend no more than one hour in a sprint review meeting for every week of the sprint — [Figure 10-3](#) shows a quick reference.



**FIGURE 10-3:** Ratio of sprint review meeting to sprint length.

Here are some guidelines for your sprint review meeting:

- » No PowerPoint slides! Show actual working functionality. Refer to the sprint backlog if you need to display a list of completed user stories.
- » The entire scrum team should participate in the meeting.
- » Anyone who is interested in the meeting may attend. The project stakeholders, the summer interns, and the CEO could all theoretically be in a sprint review. Customers may also be invited whenever available.
- » The product owner introduces the release goal, the sprint goal, and the new capabilities included.
- » The development team demonstrates what it *completed* during the sprint. Typically, the development team showcases new features or architecture.
- » The demonstration should be on equipment as close as possible to the planned production environment. For example, if you're creating a mobile application, present the features on a smartphone — perhaps hooked up to a monitor — rather than on a laptop.
- » The stakeholders can ask questions and provide feedback on the demonstrated product.
- » No non-disclosed rigged functionality, such as hard-coded values and other programming shortcuts that make the application look more mature than it currently is. Rigged functionality creates more overhead work for

the scrum team in future sprints to catch up to what the stakeholders think already exists.

- » The product owner can lead a discussion about what is coming next based on the features just presented and new items that have been added to the product backlog during the current sprint.



**REMEMBER** By the time you get to the sprint review, the product owner has already seen the functionality for each of the user stories that are going to be presented and has agreed that they are complete.

The sprint review meeting is valuable to the development team. The sprint review provides an opportunity for the development team to show its work directly. The meeting allows the stakeholders to recognize the development team for its efforts. The meeting contributes to development team morale, keeping the team motivated to try and produce ever-increasing volumes of quality work. The sprint review even establishes a certain level of friendly comparative competition between scrum teams that keeps everyone focused.



**WARNING** Sometimes, healthy competition can result in developers trying to create the coolest features or exceed the requirements of a user story — an issue known as *gold plating*. A tenet of agility is to produce only what a user story needs to pass the acceptance test. There is a risk that development team members will go beyond requirement needs in their enthusiasm, essentially wasting time that should be spent on useful product functionality. The product owner should be watchful for this. Gold-plating can be identified and avoided on a daily basis at the daily scrum or as the development team seeks clarification from the product owner.

Next, you see how to note and use the stakeholders' feedback during the sprint review meeting.

## ***Collecting feedback in the sprint review meeting***

Gather sprint review feedback informally. The product owner or scrum

master can take notes on behalf of the development team, as team members often will be engaged in the presentation and resulting conversation.

Keep in mind the example project we use throughout the book: a mobile application for XYZ Bank. Stakeholders responding to functionality they saw for the XYZ Bank mobile application might have comments such as the following:

- » From a person in sales or marketing: “You might want to consider letting the customers save their preferences, based on the results you showed. It will make for a more personalized experience going forward.”
- » From a functional director or manager: “Given what I’ve seen, you might be able to leverage some of the code modules that were developed for the ABC project last year. They needed to do similar data manipulation.”
- » From someone who works with the quality or user experience professionals in the company: “I noticed your logins were pretty straightforward; will the application be able to handle special characters?”

New user stories may come out of the sprint review. These new user stories might be new features or changes to the existing functionality.



TIP In the first few sprint reviews, the scrum master may need to remind stakeholders about agile practices. Some people hear the word “demonstration” and immediately expect fancy slides and printouts. The scrum master has a responsibility to manage these expectations and uphold agile values and practices.

The product owner needs to add any new user stories to the product backlog and order those stories by priority. The product owner also adds back into the product backlog any stories that were scheduled for the current sprint but not completed, and reorders those stories based on the most recent priorities.

The product owner needs to complete updates to the product backlog in time for the next sprint planning meeting.

When the sprint review is over, it is time for the sprint retrospective.



**TIP** You may want to take a brief break between the sprint review and the sprint retrospective so that scrum team members can come to the retrospective discussion fresh and relaxed.

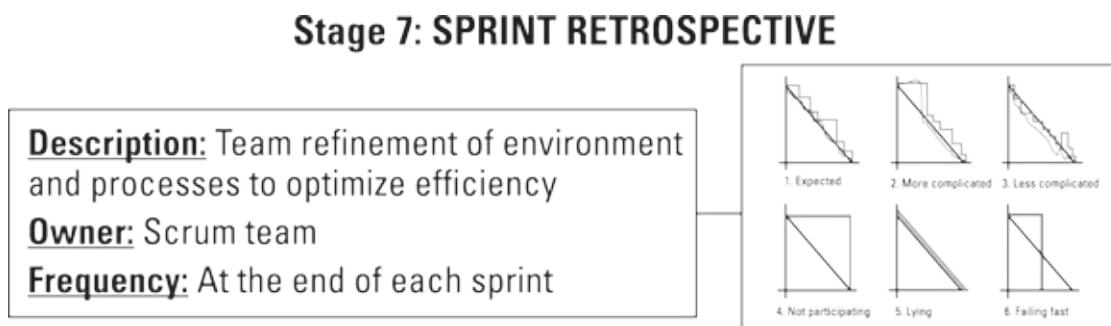
Having just completed the sprint review, the scrum team will come into the retrospective ready to inspect its processes and will have ideas for adaptation.

## The Sprint Retrospective

The *sprint retrospective* is a meeting in which the product owner, development team, and scrum master discuss how the sprint went and what they can do to improve the next sprint. The scrum team should conduct this meeting in a self-directed way. If managers or supervisors attend sprint retrospectives, scrum team members will avoid being open with each other, which limits the effectiveness of the team inspecting and adapting in a self-organizing way.

If the scrum team likes, members can invite other stakeholders to attend as well. If the scrum team regularly interacts with outside stakeholders, those stakeholders' insights can be valuable.

The sprint retrospective is stage 7 in the Roadmap to Value. [Figure 10-4](#) shows how the sprint retrospective fits into an agile project.



[FIGURE 10-4:](#) The sprint retrospective in the Roadmap to Value.

The goal of the sprint retrospective is to continuously improve your processes. Improving and customizing processes according to the needs of each individual scrum team increases scrum team morale, improves efficiency, and increases *velocity* — work output. (Find details on velocity in

[Chapter 13](#).) However, what works for one team won't necessarily work for another team. Managers outside the scrum team should not dictate how all scrum teams should overcome their challenges and should instead allow them to find the best solutions for themselves.

Your sprint retrospective results may be unique for your scrum team. For example, members of one scrum team we worked with decided that they would like to come into work early and leave early, and spend summer afternoons with their families. Another team at the same organization felt that they did better work late at night, and decided to come to the office in the afternoon and work into the evenings. The result for both teams was increased morale and increased velocity.

Use the information you learn in the retrospective to review and revise your work processes and make your next sprint better.

## STOPPING THE LINE

Taiichi Ohno, who built the Toyota Production System in the 1950s and 1960s — the beginning of lean manufacturing — decentralized assembly-line management to empower line workers to make decisions. Line workers actually had a responsibility to stop the line by pushing a red button when they found a defect or a problem on the assembly line.

Traditionally, plant managers viewed stopping the line as a failure and focused on running the assembly line at capacity as many hours of the day as possible to maximize throughput. Ohno's philosophy was that by removing constraints as they occur, you proactively create a better system rather than trying to optimize your existing process.

When first introduced, the productivity of managers who implemented this practice took an initial drop because they spent more time fixing defects in the system than the managers' teams who did not adopt the practice. The old teams declared this a victory at first. However, it didn't take long until the new teams not only caught up but also began producing more quickly, more cheaply, and with fewer defects and variance than the teams who weren't making continuous improvements in their system. This process of regular and continuous improvement is what made Toyota so successful.



**REMEMBER** Agile approaches — particularly scrum — quickly reveal problems in projects. Scrum doesn't fix problems; it simply exposes them and provides a framework for inspecting and adapting exposed issues. Data from the sprint backlog shows exactly where the development team has

been slowed down. The development team talks and collaborates. All these tools and practices help reveal inefficiencies and allow the scrum team to refine practices to improve sprint after sprint. Pay attention to what gets exposed. Don't ignore it, and don't work around it.

In the following sections, you find out how to plan for a retrospective, how to run a sprint retrospective meeting, and how to use the results of each sprint retrospective to improve future sprints.

## ***Planning for sprint retrospectives***

For the first sprint retrospective, everyone on the scrum team should think about a few key questions and be ready to discuss them. What went well during the sprint? What would you change, and how?

Everyone on the scrum team may want to make a few notes beforehand, or even take notes throughout the sprint. The scrum team could keep the roadblocks from the sprint's daily scrum meetings in mind. For the second sprint retrospective forward, you can also start to compare the current sprint with prior sprints, and track progress on the improvement efforts from sprint to sprint. In [Chapter 9](#), we mention saving sprint backlogs from prior sprints; this is where they might come in handy.

If the scrum team has honestly and thoroughly thought about what went right and what could be better, it can go into the sprint retrospective ready to have a useful conversation.

## ***The sprint retrospective meeting***

The sprint retrospective meeting is an action-oriented meeting. The scrum team immediately applies what it learned in the retrospective to the next sprint.



**REMEMBER** The sprint retrospective meeting is an action-oriented meeting, not a justification meeting. If you are hearing words like "because," the conversation is moving away from action and toward rationale.

One of the rules of scrum is to spend no more than 45 minutes in a sprint retrospective meeting for every week of the sprint. [Figure 10-5](#) shows a quick reference.



**FIGURE 10-5:** Ratio of sprint retrospective meeting to sprint length.

The sprint retrospective should cover three primary questions:

- » What went well during the sprint?
- » What would we like to change?
- » How can we implement that change?

The following areas are also open for discussion:

- » **Results:** Compare the amount of work planned with what the development team completed. Review the sprint burndown chart (see [Chapter 9](#)) and what it tells the development team about how it's working.
- » **People:** Discuss team composition and alignment.
- » **Relationships:** Talk about communication, collaboration, and working in pairs.
- » **Processes:** Go over support, development, and peer review processes.
- » **Tools:** How are the different tools working for the scrum team? Think about the artifacts, electronic tools, communication tools, and technical tools.
- » **Productivity:** How can the team improve productivity and get the most work done in the next sprint?

It helps to have these discussions in a structured format. Esther Derby and Diana Larsen, authors of *Agile Retrospectives: Making Good Teams Great* (Pragmatic Bookshelf, 2006), have a great agenda for sprint retrospectives that keeps the team focused on discussions that will lead to real improvement:

- 1. Set the stage.**

Establishing the goals for the retrospective upfront will help keep your scrum team focused on providing the right kind of feedback later in the meeting. As you progress into later sprints, you may want to have retrospectives that focus on one or two specific areas for improvement.

- 2. Gather data.**

Discuss the facts about what went well in the last sprint and what needed improvement. Create an overall picture of the sprint; consider using a whiteboard to write down the input from meeting attendees.

- 3. Generate insights.**

Take a look at the information you just gathered and come up with ideas about how to make improvements for the next sprint.

- 4. Decide what to do.**

Determine — as a team — which ideas you'll put into place. Decide on specific actions you can take to make the ideas a reality.

- 5. Close the retrospective.**

Reiterate your plan of action for the next sprint. Thank people for contributing. Also find ways to make the next retrospective better!

For some scrum teams, it might be difficult to open up at first. The scrum master may need to ask specific questions to start discussions. Participating in retrospectives takes practice. What matters is to encourage the scrum team to take responsibility for the sprint — to truly embrace being self-managing.

In other scrum teams, a lot of debate and discussion ensues during the retrospective. The scrum master can find it challenging to manage these discussions and keep the meeting within its allotted time, but that is what needs to be done.

Be sure to use the results from your sprint retrospectives to inspect and adapt throughout your project.

## **Inspecting and adapting**

The sprint retrospective is one of the best opportunities you have to put the ideas of inspect and adapt into action. You came up with challenges and solutions during the retrospective. Don't leave those solutions behind after the meeting; make the improvements part of your work every day.

You could record your recommendations for improvement informally. Some scrum teams post the actions identified during the retrospective meeting in the team area to ensure visibility and action on the items listed. Many teams also add action items to the product backlog to ensure that they implement them during an upcoming sprint.



**TIP** To become more agile, scrum teams focus on small changes with big value. We like teams to take at least one improvement into each sprint, a process sometimes referred to as *scrumming the scrum*.

In subsequent sprint retrospective meetings, it's important to review the evaluations of the prior sprint and make sure you put the suggested improvements into place.

# Chapter 11

## Preparing for Release

---

### IN THIS CHAPTER

- » Getting your product ready to ship
- » Organizing for operational support
- » Preparing the rest of the organization for the release
- » Making sure the marketplace is ready for your release

Releasing new product features to customers has a special set of challenges. The development team has specific tasks for a product release that differ from the tasks involved with creating functionality during normal sprints. The organization sponsoring the product may need to prepare to support the product. You want customers to be able to correctly use the released product.

This chapter covers how to manage the final sprint, if needed — the release sprint — before product release. You also discover how to prepare your organization and the marketplace for the product release.

### *Preparing the Product for Deployment: The Release Sprint*

The work that takes place during regular development sprints should be whole and complete, including testing and technical documentation, before you demonstrate your product. The product of a development sprint is working functionality.

However, there may be activities, not related to creating product features, that the development team can't realistically complete within development sprints and that might even introduce unacceptable overhead. To accommodate prerelease activities and help ensure that the release goes well, scrum teams may schedule a release sprint as the final sprint prior to releasing

functionality to customers.



**REMEMBER** If a scrum team requires a release sprint, it probably means that the broader organization can't support it, which is an anti-pattern to becoming agile. Every type of work or activity required to release functionality to the market should be part of the sprint-level definition of done. That is the goal for agile teams.

The release sprint contains only those things needed to move working functionality to the marketplace. The following are examples of release sprint backlog items. See if you can think of possible ways to shift any of these into the development sprint:

- » Creating user documentation for the most recent version of the product
- » Performance testing, load testing, security testing, and any other checks to ensure that the working software will perform acceptably in production
- » Integrating the product with enterprise-wide systems, where testing might take days or weeks
- » Completing organizational or regulatory procedures that are mandatory prior to release
- » Preparing release notes — final notes about changes to the product
- » Preparing the deployment package, enabling all the code for the product features to move to production at one time
- » Deploying your code to the production environment

## UNDERSTANDING THE ROLE OF DOCUMENTATION

What's the difference between the technical documentation that you create during a sprint and the user documentation that you may create in your release sprint?

Your *technical documentation* should be barely sufficient, with no frills and just enough information to tell the development team — and perhaps future development teams — how to create and update the product. If, on the last day of the sprint, the development team won the lottery and retired to Costa Rica, a new development team should be able to review the technical documentation and easily pick up where the former team left off.

Your *user documentation* tells your customers how to use your product. You may want user documentation crafted specifically for each of your customers. For example, a mobile banking application might need a frequently asked questions (FAQs) section for banking customers. The same application might have a feature that enables marketing managers to upload ad messages to the application; you'd want to make sure those managers have instructions for the upload feature as well. Because your product will have changes throughout each sprint of the release, it might be more efficient to wait until the last responsible minute to create your user documentation, after the stakeholders agree that the functionality is ready for release.

Some aspects of a release sprint are different from a development sprint:

- » You do not develop any new functionality requirements from the product backlog. Although you have functionality freeze, you do not have code freeze because the development team will need to make adjustments to respond to feedback from release sprint activities, such as performance testing or focus groups.
- » Based on the work you need to do, your release sprint may be a different length than your regular development sprints. In addition, you won't have the concept of velocity because you won't be doing the same type of work that you do in development sprints.
- » The definition of done is different for work completed during a release sprint. In a development sprint, *done* means the completion of working functionality for a user story. In a release sprint, the definition is the completion of all tasks required for release.
- » A release sprint includes tests and approvals that may not be practical to do in a development sprint, such as performance testing, load testing, security testing, focus groups, and legal review.



**REMEMBER** Agile development teams may create two definitions of done: one for sprints and one for releases.

[Table 11-1](#) shows a comparison between the activities of a development sprint and a release sprint. For detailed descriptions of the key elements in a sprint, see [Chapters 8 through 10](#).

## **TABLE 11-1 Development Sprint Elements versus**

# Release Sprint Elements

Element	Used in Development Sprint	Used in Release Sprint
Sprint planning	Yes	Yes
Product backlog	Yes	No
Sprint backlog	Yes For a development sprint, your sprint backlog contains user stories and the tasks needed to create each user story. You estimate user stories relatively, with story points. (See <a href="#">Chapters 7</a> and <a href="#">8</a> .)	Yes In a release sprint, you no longer need to put your requirements in the user story format. Instead, you create only a list of tasks needed for the release. You also do not use story points. Instead, add the estimated hours each task will take when planning the release sprint, in the same way you break down and estimate tasks during the development sprint planning.
Burndown chart	Yes	Yes
Daily scrum	Yes	Yes Involve stakeholders from outside the scrum team who have tasks associated with releasing the product, such as enterprise build managers or other configuration managers.
Daily activities	In a development sprint, your daily activities focus on creating shippable functionality.	In a release sprint, your daily activities focus on preparing your working functionality for external release.
End-of-day reporting	Yes	Yes
Sprint review	Yes	Yes Some organizations use a release sprint review as a <i>go or no-go meeting</i> to authorize launching the functionality.
Sprint retrospective	Yes	Yes This can be an opportunity to inspect the entire sprint and plan for adapting in the next release.



**WARNING** A release sprint should not be a parking lot for tasks that the development team didn't finish in the development sprints. You may not be surprised to hear that development teams are sometimes tempted to delay tasks until the release sprint. You can avoid this by ensuring that the scrum team has created a proper definition of done for requirements in development sprints, including testing, integration, and documentation.

While running a release sprint, you also need to prepare your organization for the product release. The next sections discuss how to prepare for supporting the new functionality in the marketplace and how to get stakeholders in your company or organization ready for product deployment.

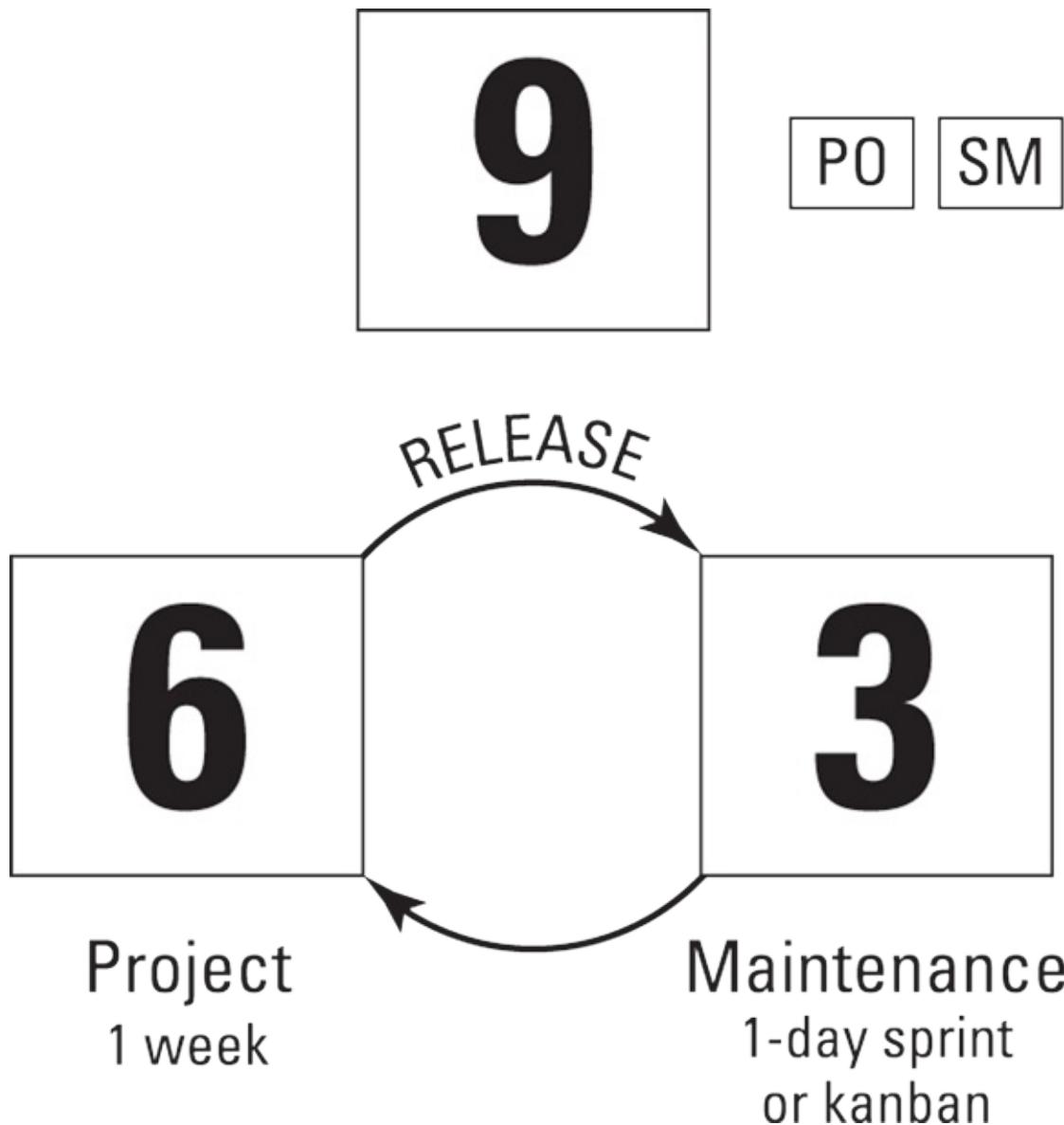
## ***Preparing for Operational Support***

After your product is released to the customer, someone will have to support it. Supporting your product involves responding to customer inquiries, maintaining the system in a production environment, and enhancing existing functionality to fill minor gaps. Although new development work and operational support work are both important, they involve different approaches and cadences.

Separating new development and support work ensures that new development teams can focus on continuing to bring innovative solutions to customers at a faster rate than if the team frequently switches between the two types of work.

A scrum team doing new development can plan and develop new working functionality within a one-to two-week sprint, but it's difficult to plan when operational or maintenance issues will arise. Maintenance work usually requires shorter timeboxed iterations, typically no more than one day, which is usually the longest the organization can go without changing priorities with any issues in production.

We recommend a model that separates new development and maintenance work, as illustrated in [Figure 11-1](#).



**FIGURE 11-1:** Operational support scrum team model.

For a scrum team of nine developers, for instance, we would divide the development team into two teams, one with six developers, and the other with three. (These numbers are flexible.) The team of six does new development project work from the product backlog in one-week to two-week sprints, as described in [Chapters 7 through 10](#). The work that the team commits to during the sprint planning meeting will be the only work they do.

The team of three are our firefighters and do maintenance and support work in one-day sprints or by using kanban. (You learn about kanban in [Chapter 4](#).) Single-day sprints allow the scrum team to triage all incoming requests from

the previous day, plan the highest-priority items, implement those items as a team, and review the results at the end of the day (or even earlier) for go or no-go approval before pushing the changes to production. For continuity, the product owner and scrum master are the same for each team.



**REMEMBER** Although the newly modified project development team is smaller than before, there are still enough developers to keep new development efforts moving forward, uninterrupted by maintenance work. By the time you begin releasing functionality to the market, your scrum team will be working well together and the developers will have increased their versatility by being able to complete more types of tasks than when the project first started.

The project development team will have periodic releases to production, such as once every 90 days. At each release, one developer will rotate to the maintenance team, armed with first-hand knowledge of the functionality being deployed to production. At the same time, one developer from the maintenance team will be rotated into the project development team, equipped with first-hand knowledge of what it's like to support the product in the real world. This rotation continues at each release.



**TECHNICAL STUFF** *Development Operations (DevOps)* is the collaboration and integration between software developers and IT operations (which includes functions such as systems administration and server maintenance). Taking a DevOps approach enables developers and operations to work together to tighten cycle times of deployment.

This DevOps model ensures that everyone gets to do new product development as well as maintenance work, and that product knowledge is continually shared effectively between the two development teams. This approach improves DevOps and facilitates cross-functional team members. It also minimizes any disruption the teams may experience from changing team members because the rotations happen only at each release rather than daily or weekly.

When preparing for release, establishing expectations upfront of how the functionality will be supported in production allows the scrum team to develop the product in a way that enables the team to effectively support the product after it is deployed. It increases ownership across the scrum team and heightens the team's awareness and dedication to long-term success.

## ONE-DAY SPRINTS

We recommend running one-day sprints for maintenance teams. By framing each day in the sprint cycle, the scrum team operates in a solid feedback loop, ensuring continuous inspection and adaptation as well as regular stakeholder involvement.

By using the same formulas for timeboxing scrum events, you won't spend hours in planning or reviews as you would with one-to four-week sprints. Dividing one-week scrum event timeboxes by five days means you spend about 25 minutes triaging the maintenance product backlog and planning the day, about 12-15 minutes for the sprint review to determine go or no-go to production, and an additional 10 minutes to inspect and adapt the team's processes and identify any issues that should be continued or discontinued the next day.

The key to operating in one-day sprints is making sure maintenance items are broken down small enough so that developers can complete them in less than a day. This approach ensures that customers get something every day rather than wait for weeks.

## *Preparing the Organization for Product Deployment*

A product release often affects a number of departments in a company or organization. To get the organization ready for the new product, the product owner and scrum master need to add items relevant to the organization to the *release sprint backlog*. (See how to create a sprint backlog in [Chapter 8](#).)

The release sprint backlog should also cover activities for the development team. It also needs to address activities to be performed by groups in the organization but outside the scrum team to prepare for the product deployment. These departments might include the following:

- » **Marketing:** Do marketing campaigns related to the new product need to launch at the same time as the product?
- » **Sales:** Are there specific customers who need to know about the product?

Will the new product cause an increase in sales?

- » **Logistics:** Is the product a physical item that includes packaging or shipping?
- » **Product support:** Does the customer service group have the information it needs to answer questions about the new product? Will this group have enough people on hand in case customer questions increase when the product launches?
- » **Legal:** Does the product meet legal standards, including pricing, licensing, and correct verbiage, for release to the public?

The departments that need to be ready for the product launch and the specific tasks these groups need to complete will, of course, vary from organization to organization. A key to release success, however, is that the product owner and scrum master involve the right people and ensure that those people clearly understand what they need to do to be ready for the functionality release.

As with development sprints, in your release sprint, you can effectively use daily scrums, sprint review meetings, and sprint retrospectives with department colleagues involved in preparing for product deployment. You can even use a task board, like the one we describe in [Chapter 9](#).

During your release sprint, you also need to include one more group in your planning: the product customer. The next section discusses getting the marketplace ready for your product.

## ***Preparing the Marketplace for Product Deployment***

The product owner is responsible for working with other departments to ensure that the marketplace — existing customers and potential customers — is ready for what's coming. The marketing or sales teams may lead this effort; team members look to the product owner to keep them informed on the release date and the features that will be part of the release.



**REMEMBER** Some software products are only for internal employee use. Certain things you’re reading in this section might seem like overkill for an internal application — an application released only within your company. However, many of these steps are still good guidelines for promoting internal applications. Preparing customers, whether internal or external, for new products can be a key part of product success.

To help prepare customers for the product release, the product owner may want to work with different teams to ensure the following:

- » **Marketing support:** Whether you’re dealing with a new product or new features for an existing product, the marketing department should leverage the excitement of the new product functionality to help promote the product and the organization.
- » **Customer testing:** If possible, work with your customers (some people use focus groups) to get real-world feedback about the product from a subset of end users. Your marketing team can also use this feedback translate into testimonials for promoting the product right away.
- » **Marketing materials:** An organization’s marketing group also prepares the promotional and advertising plans, as well as packaging for physical media. Media materials, such as press releases and information for analysts, need to be ready, as do marketing and sales materials.
- » **Support channels:** Ensure that customers understand the available support channels in case they have questions about the product.

Review the tasks on your release sprint backlog from the customer’s standpoint. Think of the personas you used when creating your user stories. Do those personas need to know something about the product? Update your launch checklist with items that would be valuable to customers represented by your personas. You can find more information about personas in [Chapter 8](#).

Finally, you’re there — release day. Whatever role you played along the way, this is the day you worked hard to achieve. It’s time to celebrate!

## Part 4

# Agile Management

## **IN THIS PART ...**

- Respond effectively to changes in scope.
- Manage vendors and contracts for success.
- Monitor and adjust schedules and budget.
- Self-organize for optimal communications.
- Inspect and adapt to increase quality and mitigate risk.

# Chapter 12

# Managing Scope and Procurement

---

## IN THIS CHAPTER

- » Finding out how scope management is different on agile projects
- » Managing scope and scope changes with agile processes
- » Seeing the different approach agile processes bring to procurement
- » Managing procurement on agile projects

Scope management is part of every project. To create a product, you have to understand basic product requirements and the work it will take to fulfill those requirements. You need to be able to prioritize and manage scope changes as new requirements arise. You have to verify that finished product features fulfill customers' needs.

Procurement is also part of many projects. If you need to look outside your organization for help completing a project, you should know how to go about procuring goods and services. You will want to know how to collaborate with vendor teams during the project. You should also know something about creating contracts and different cost structures.

In this chapter, you find out how to manage scope in an agile project and take advantage of agile methods' welcoming approach to informed change. You also find out how to manage procurement of goods and services to deliver product scope on an agile project. First, we review traditional scope management.

## *What's Different about Agile Scope Management?*

Historically, a large part of project management is scope management. *Product scope* is all the features and requirements that a product includes. *Project scope* is all the work involved in creating a product.

Traditional project management treats changing requirements as a sign of failure in upfront planning. Agile projects, however, have variable scope so that project teams can immediately and incrementally incorporate learning and feedback, and ultimately create better products. The signers of the Agile Manifesto recognized that scope change is natural and beneficial. Agile approaches specifically embrace change and use it to make better-informed decisions and more useful products.



**TIP** If you run an agile project and your requirements don't change because you learned nothing along the way, that is a failure. Your product backlog should change often as you learn from stakeholder and customer feedback. It's unlikely that you knew everything at the beginning of the project.



**REMEMBER** [Chapter 2](#) details the Agile Manifesto and the 12 Agile Principles. (If you haven't yet checked out that chapter, flip back to it now. We'll wait.) The manifesto and the principles answer the question, "How agile are we?" The degree to which your project approach supports the manifesto and the principles helps determine how agile your methods are.

The agile principles that relate the most to scope management follow:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
10. Simplicity — the art of maximizing the amount of work not done — is

essential.

Agile approaches to scope management are fundamentally different than traditional methods for scope management. Consider the differences you see in [Table 12-1](#).

**TABLE 12-1** Traditional versus Agile Scope Management

<i>Scope Management with Traditional Approaches</i>	<i>Scope Management with Agile Approaches</i>
Project teams attempt to identify and document complete scope at the beginning of the project, when the teams are the least informed about the product.	The product owner gathers high-level requirements at the beginning of the project, breaking down and further detailing requirements that are going to be implemented in the immediate future. Requirements are gathered and refined throughout the project as the team's knowledge of customer needs and project realities grows.
Organizations view scope change after the requirements phase is complete as negative.	Organizations view change as a positive way to improve a product as the project progresses. Changes late in the project, when you know the most about the product, are often the most valuable changes.
Project managers rigidly control and discourage changes after stakeholders sign off on requirements.	Change management is an inherent part of agile processes. You assess scope and have an opportunity to include new requirements with every sprint. The product owner determines the value and priority of new requirements and adds those requirements to the product backlog.
The cost of change increases over time, while the ability to make changes decreases.	You fix resources and schedule initially. New features with high priority don't necessarily cause budget or schedule slip; they simply push out the lowest-priority features. Iterative development allows for changes with each new sprint.
Projects often include scope <i>bloat</i> , unnecessary product features included out of fear of mid-project change.	The scrum team determines scope by considering which features directly support the product vision, the release goal, and the sprint goal. The development team creates the most valuable features first to guarantee their inclusion and to ship those features as soon as possible. Less valuable features might never be created, which may be acceptable to the business and the customer after they have the highest-value features.

At any point in an agile project, anyone — the scrum team, stakeholders, or anyone else in the organization with a good idea — can identify new product requirements. The product owner determines the value and priority of new requirements and prioritizes those requirements against other requirements in the product backlog.



TECHNICAL STUFF

Traditional project management has a term to describe requirements that change after the project's initial definition phase: *scope creep*. Waterfall doesn't have a positive way to incorporate changes mid-project, so scope changes often cause large problems with a waterfall project's schedule and budget. (For more on the waterfall methodology, see [Chapter 1](#).) Mention "scope creep" to a seasoned project manager, and you might even see him or her shudder.

During sprint planning at the beginning of each sprint, the scrum team can use the product backlog priority to help decide whether a new requirement should be part of the sprint. Lower-priority requirements stay in the product backlog for future consideration. You can read about planning sprints in [Chapter 8](#).

The next section addresses how to manage scope on an agile project.

## Managing Agile Scope

Welcoming scope change helps you create the best product possible. Embracing change, however, requires that you understand the current scope and know how to deal with updates as they arise. Luckily, agile approaches have straightforward ways to manage new and existing requirements:

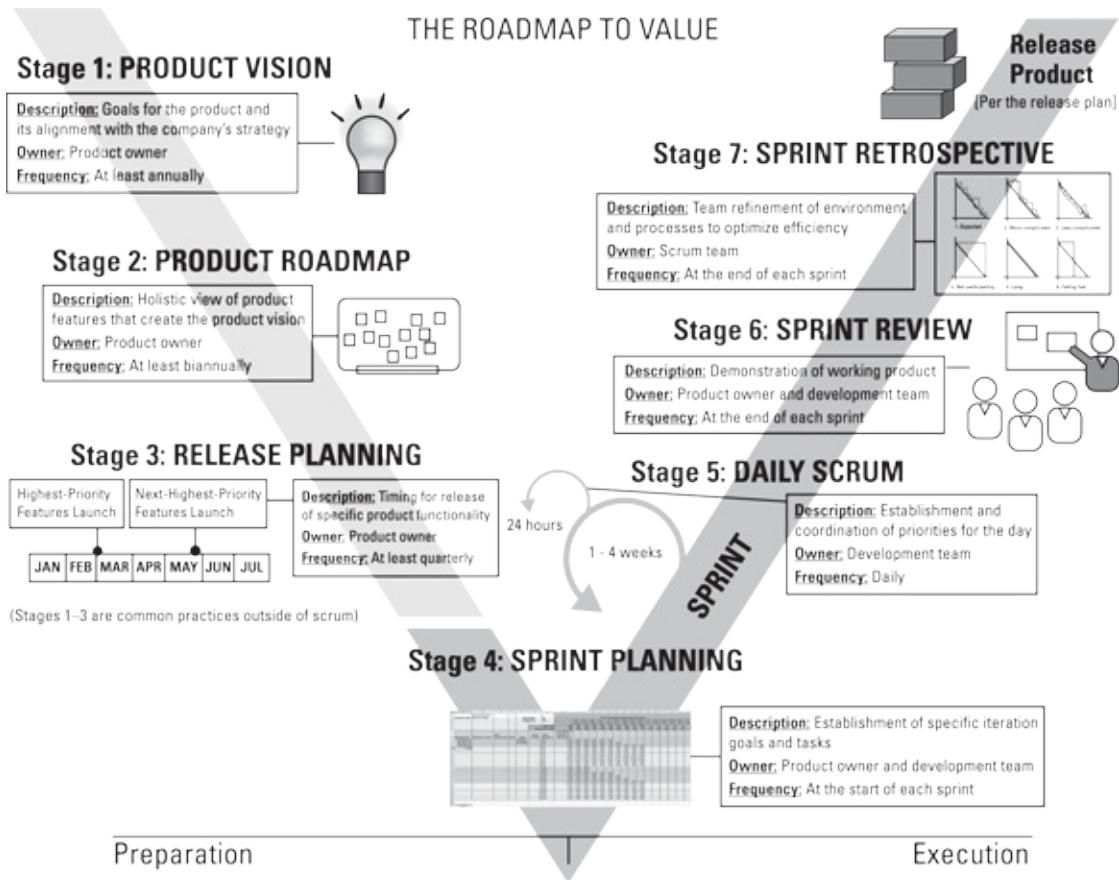
- » The product owner ensures that the rest of the project team — the scrum team plus the project stakeholders — clearly understands the existing scope for the project, the product vision, the current release goal, and the current sprint goal.
- » The product owner determines the value and priority of new requirements in relation to the product vision, release goal, sprint goals, and existing requirements.
- » The development team creates product requirements in order of priority to release the most important parts of the product first.

In the following sections, you find out how to understand and convey scope in different parts of an agile project. You see how to evaluate priorities as

new requirements arise. You also find out how to use the product backlog and other agile artifacts to manage scope.

## ***Understanding scope throughout the project***

At each stage in an agile project, the scrum team manages scope in different ways. A good way to look at scope management throughout a project is by using the Roadmap to Value, first presented in [Chapter 7](#) and shown again in [Figure 12-1](#).



## **FIGURE 12-1:** The Roadmap to Value.

Consider each part of the Roadmap to Value:

- » **Stage 1, product vision:** The product vision statement establishes the outer boundary of the functionality that the product will include, and is the first step in establishing project scope. The product owner is responsible for ensuring that all members of the project team know the product vision statement and that everyone on the project team interprets the statement correctly.

- » **Stage 2, product roadmap:** During product roadmap creation, the product owner refers to the vision statement and ensures that features support the vision statement. As new features materialize, the product owner needs to understand them and be able to clearly communicate to the development team and stakeholders the scope of these features and how they support the product vision.
- » **Stage 3, release planning:** During release planning, the product owner needs to determine a release goal — the midterm boundary of functionality that is planned to go to market at the next release — and select only the scope that supports that release goal.
- » **Stage 4, sprint planning:** During sprint planning, the product owner needs to ensure that the scrum team understands the release goal and plans each sprint goal — the immediate boundary of functionality to be potentially shippable at the end of the sprint — based on that release goal. The product owner and development team select only the scope that supports the sprint goal as part of the sprint. The product owner will also ensure that the development team understands the scope of the individual user stories selected for the sprint.
- » **Stage 5, daily scrum:** The daily scrum meeting can be a launching point for scope change for future sprints. The daily scrum meeting is a focused, 15-minute meeting for the development team to state three things: the preceding day's completed work, the scope of work for the coming day, and any roadblocks the development team may have. However, the three subjects of the daily scrum often reveal larger opportunities for scope changes.

When topics come up that warrant a bigger discussion than the time and format of the daily scrum meeting allows, a scrum team can decide to have an after-party meeting. In the after-party, scrum team members talk about issues affecting their progress toward the sprint goal. If opportunities for new functionality — new scope — are identified during the sprint, the product owner evaluates them and may add and prioritize them on the product backlog for a future sprint.

- » **Stage 6, sprint review:** The product owner sets the tone of each sprint review meeting by reiterating the scope of the sprint — the sprint goal that the scrum team pursued and what was completed. Especially during

the first sprint review, it's important that the stakeholders in the meeting have the right expectations about scope.

Sprint reviews can be inspiring. When the entire project team is in one room, interacting with the working product, members may look at the product in new ways and come up with ideas to improve the product. The product owner updates the product backlog with new scope based on feedback received in the sprint review.

- » **Stage 7, sprint retrospective:** In the sprint retrospective, the scrum team members can discuss how well they met the scope commitments they made at the beginning of the sprint. If the development team was not able to achieve the sprint goal identified during sprint planning, its members will need to refine planning and work processes to make sure they can select the right amount of work for each sprint. If the development team met its goals, it can use the sprint retrospective to come up with ways to add more scope to future sprints. Scrum teams aim to improve productivity with every sprint.

## *Introducing scope changes*

Many people, even people outside the organization, can suggest a new product feature on an agile project. You might see new ideas for features from the following:

- » User community feedback, including groups or people who are given an opportunity to preview the product
- » Business stakeholders who see a new market opportunity or threat
- » Executives and senior managers who have insight into long-term organizational strategies and changes
- » The development team, which is learning more about the product every day, and is closest to the working product
- » The scrum master, who may find an opportunity while working with external departments or clearing development team roadblocks
- » The product owner, who often knows the most about the product and the stakeholders' needs

Because you will receive suggestions for product changes throughout an agile

project, you want to determine which changes are valid and manage the updates. Read on to see how.

## Managing scope changes

When you get new requirements, use the following steps to evaluate and prioritize the requirements and update the product backlog.



**WARNING** Do not add new requirements to sprints already in progress, unless the development team requests them, usually due to unexpected increased capacity.

- 1. Assess whether the new requirement should be part of the product, the release, or the sprint by asking some key questions about the requirement:**

- a. Does the new requirement support the product vision statement?**

- If yes, add the requirement to the product backlog and product roadmap.
- If no, the requirement shouldn't be part of the project. It may be a good candidate for a separate project.

- b. If the new requirement supports the product vision, does the new requirement support the current release goal?**

- If yes, the requirement is a candidate for the current release plan.
- If no, leave the requirement on the product backlog for a future release.

- c. If the new requirement supports the release goal, does the new requirement support the current sprint goal?**

- If yes and if the sprint has not started, the requirement is a candidate for the current sprint backlog.
- If no or the sprint has already started or both, leave the requirement on the product backlog for a future sprint.

- 2. Estimate the effort for the new requirement.**

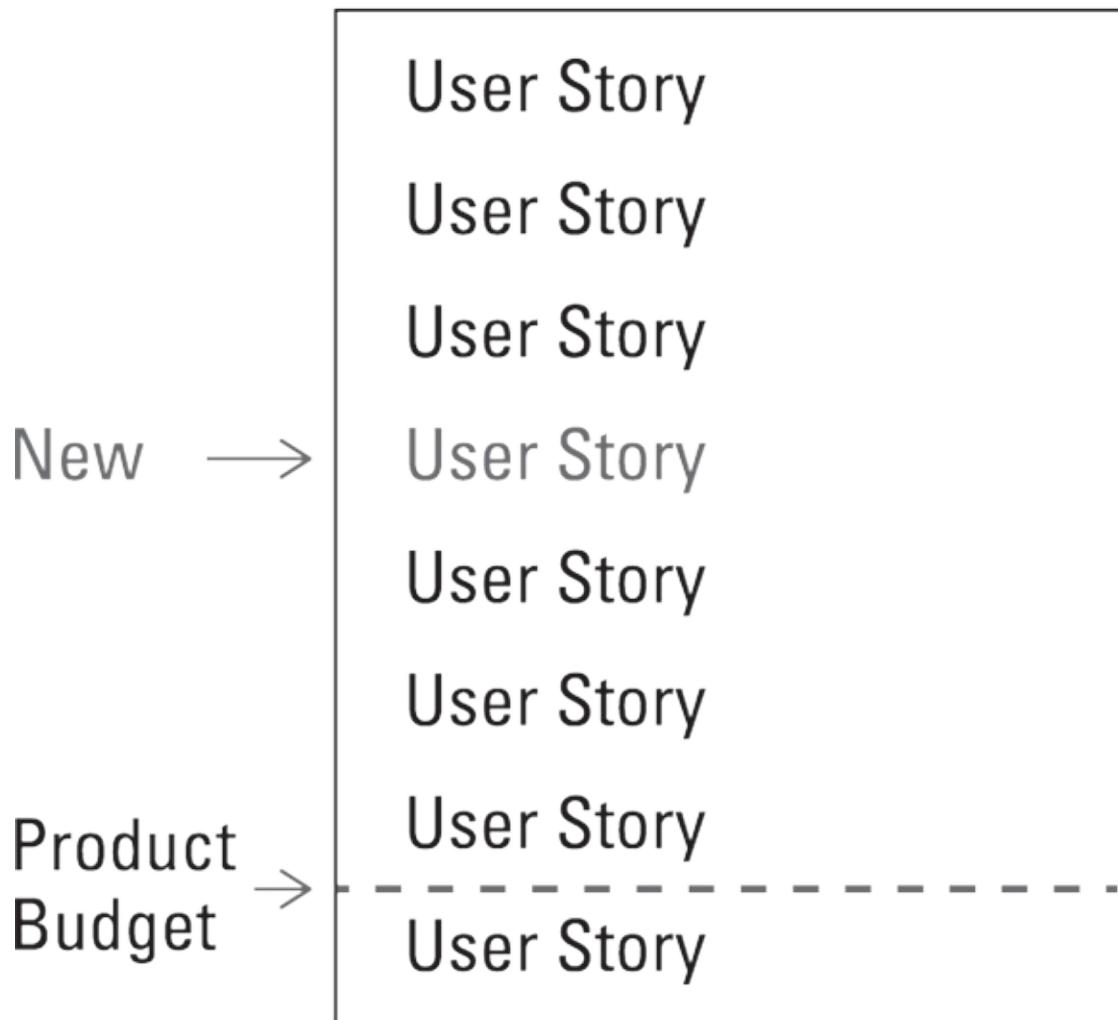
The development team estimates the effort. Find out how to estimate requirements in [Chapter 7](#).

**3. Prioritize the requirement against other requirements in the product backlog and add the new requirement to the product backlog, in order of priority.**

Consider the following:

- The product owner knows the most about the product's business needs and how important the new requirement may be in relation to other requirements. The product owner may also reach out to project stakeholders for additional insight to a requirement's priority.
- The development team may also have technical insight about a new requirement's priority. For example, if Requirement A and Requirement B have equal business value, but you need to complete Requirement B for Requirement A to be feasible, the development team will need to alert the product owner. Requirement B may need to be completed first.
- Although the development team and project stakeholders can provide information to help prioritize a requirement, determining priority is ultimately the product owner's decision.
- Adding new requirements to the product backlog may mean other requirements move down the list in the product backlog. [Figure 12-2](#) shows the addition of a new requirement in the product backlog.

## Remaining Product Backlog



**FIGURE 12-2:** Adding a new requirement to the product backlog.

The product backlog is a complete list of all known scope for the product and is your most important tool for managing scope change on an agile project.

Keeping the product backlog up to date will allow you to quickly prioritize and add new requirements. With a current product backlog, you always understand the scope left in a project. [Chapter 7](#) has more information about prioritizing requirements.

### ***Using agile artifacts for scope management***

From the vision statement through the sprint plan, all the artifacts in agile project management support you in your scope management efforts. Progressively decompose, or break down, requirements as features rise to the

top of the priority list. We talk about decomposition and progressive elaboration of requirements in [Chapter 7](#).

[Table 12-2](#) reveals how each agile artifact, including the product backlog, contributes to ongoing scope refinement.

**TABLE 12-2 Agile Artifacts and Scope Management Roles**

<i>Artifact</i>	<i>Role in Establishing Scope</i>	<i>Role in Scope Change</i>
<b>Vision statement:</b> A definition of the product's end goal. <a href="#">Chapter 7</a> has more about the vision statement.	Use the vision statement as a benchmark to judge whether features belong in the scope for the current project.	When someone introduces new requirements, those requirements must support the product vision statement.
<b>Product roadmap:</b> A holistic view of product features that create the product vision. <a href="#">Chapter 7</a> has more about the product roadmap.	Product scope is part of the product roadmap. Requirements at a feature level are good for business conversations about what it means to realize the product vision.	Update the product roadmap as new requirements arise. The product roadmap provides visual communication of the new feature's inclusion in the project.
<b>Release plan:</b> A digestible midterm target focused around a minimum set of marketable features. <a href="#">Chapter 8</a> has more about the release plan.	The release plan shows the scope of the current release. You may want to plan your releases by themes — logical groups of requirements.	Add new features that belong in the current release to the release plan. If the new user story doesn't belong in the current release, leave it on the product backlog for a future release.
<b>Product backlog:</b> A complete list of all known scope for the product. <a href="#">Chapters 7 and 8</a> offer more about the product backlog.	If a requirement is in the scope of the product vision, it is part of the product backlog.	The product backlog contains all scope changes. New, high-priority features push lower-priority features down on the product backlog.
<b>Sprint backlog:</b> The user stories and tasks in the scope of the current sprint. <a href="#">Chapter 8</a> has more about the sprint backlog.	The sprint backlog contains the user stories that are in scope for the current sprint.	The sprint backlog establishes what is allowed in the sprint. After the development team commits to the sprint goal in the sprint-planning meeting, only the development team can modify the sprint backlog.

## *What's Different about Agile Procurement?*

Another part of project management is *procurement*, managing the purchase of services or goods needed to deliver the product's scope. Like scope, procurement is part of the investment side of a project.

[Chapter 2](#) explains that the Agile Manifesto values *customer collaboration over contract negotiation*. This sets an important tone for procurement relationships on agile projects.

Valuing customer collaboration more than contract negotiation doesn't mean that agile projects have no contracts: Contracts and negotiation are critical to business relationships. However, the Agile Manifesto sets forth the idea that a buyer and seller should work together to create products, and that the relationship between the two is more important than quibbling over ill-informed details and checking off contract items that may or may not ultimately be valuable to customers.

All 12 Agile Principles apply to procurement on agile projects. However, the following seem to stand out the most when securing goods and services for an agile project:

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
10. Simplicity — the art of maximizing the amount of work not done — is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.

[Table 12-3](#) highlights the differences between procurement on traditional projects and procurement on agile projects.

## **TABLE 12-3 Traditional versus Agile Procurement Management**

<i>Procurement Management with Traditional Approaches</i>	<i>Procurement Management with Agile Approaches</i>
	The self-managing development team plays a larger

The project manager and the organization are responsible for procurement activities.

part in identifying items needing procurement. The scrum master facilitates the acquisition of needed items for the development team.

Contracts with service providers often include provisions for fixed requirements, extensive documentation, a comprehensive project plan, and other traditional deliverables based on a waterfall lifecycle.

Contracts for agile projects are based on the evaluation of working functionality at the end of each sprint, not on fixed deliverables and documentation that may or may not contribute to delivering quality products.

Contract negotiation between buyers and sellers can sometimes be challenging. Because negotiation is often a stressful activity, it can damage the relationship between the buyer and the seller before work even starts on a project.

Agile project teams focus on keeping a positive, cooperative relationship between buyers and sellers from the start of the procurement process.

Switching vendors after a project starts can be costly and time-consuming because a new vendor must try to understand the old vendor's massive amount of work in progress.

Vendors provide completed, working functionality at the end of each sprint. If vendors change mid-project, the new vendor can immediately start developing requirements for the next sprint, avoiding a long, costly transition.



TECHNICAL STUFF

Both waterfall and agile project teams are interested in vendor success. Traditional project approaches were firm in their accountability for compliance, defining success as checking off documents and deliverables in a list. Agile project approaches, by contrast, are firm in their accountability for end results, defining success with working functionality.

The next section shows how to manage procurement on agile projects.

## Managing Agile Procurement

This section focuses on how agile project teams go through the procurement process: from determining need, selecting a vendor, and creating a contract through working with a vendor and closing out the contract at the end of a buyer-seller project.

### Determining need and selecting a vendor

On agile projects, procurement starts when the development team decides it needs a tool or the services of a third-party to create the product.



**REMEMBER** Agile project development teams are self-managing and self-organizing, and they get to make the decisions about what is best for maximizing development output. Self-management applies to all project management areas, including procurement. Find out more about self-managing teams in [Chapters 6](#) and [14](#).

Development teams have a number of opportunities to consider outside goods and services:

- » **Product vision stage:** The development team may start thinking about the tools and skills necessary to help reach the product vision. At this stage, it may be prudent to research needs but not begin the purchase process.
- » **Product roadmap stage:** The development team starts to see specific features to create and may know some of the goods or services necessary to help create the product.
- » **Release planning:** The development team knows more about the product and can identify specific goods or services that will help meet the next release goal.
- » **Sprint planning:** The development team is in the trenches of development and may identify urgent needs for the sprint.
- » **Daily scrum:** Development team members state impediments. Procuring goods or services may help remove these impediments.
- » **Throughout the day:** Development team members communicate with one another and collaborate on tasks. Specific needs may arise from the development team's conversations.
- » **Sprint review meeting:** Project stakeholders may identify new requirements for future sprints that warrant procurement of goods or services.
- » **Sprint retrospective:** The development team may discuss how having a specific tool or service could have helped the past sprint and suggest a purchase for future sprints.



**TIP** Sometimes you can find the goods or services you need for a project in your organization. Before looking at buying an item or working with vendors, the scrum master determines whether the tool or the person with the skills to fulfill the services the development team needs is available internally. If internal resources or people can meet the development team's needs, the scrum team saves money.

After the development team determines it needs a good or service, the development team and the scrum master work with the product owner to procure any necessary funds. The product owner is responsible for managing project scope against the project budget, so the product owner is ultimately responsible for any project purchases. The scrum master usually manages the vendor relationship on behalf of the scrum team after procurement is initiated with the vendor.

When procuring goods, the development team may need to compare tools and vendors before deciding on a purchase. When procuring goods, after you choose what to buy and where to get it, the process is usually straightforward: Make the purchase and take delivery.

Procuring services is usually a longer and more complex process than purchasing goods. Some agile-specific considerations for selecting a services vendor include the following:

- » Whether the vendor can work in an agile project environment and, if so, how much experience the vendor has with agile projects
- » Whether the vendor can work on-site with the development team
- » Whether the relationship between the vendor and the scrum team is likely to be positive and collaborative



**WARNING** The organization or company you work for may be subject to laws and regulations for choosing vendors. Companies involved in government work, for example, often need to gather multiple proposals

and bids from companies for work that will cost more than a certain amount of money. Although your cousin or your friend from college might be the most qualified person to complete the work, you may run into trouble if you don't follow applicable laws. Check with your company's legal department if you're in doubt about how to streamline bloated processes.

After you choose a service vendor, you need to create a contract so that the vendor can start work. The next section explains how contracts work on agile projects.

## ***Understanding cost approaches and contracts for services***

After the development team and the product owner have chosen a vendor, they need a contract to ensure agreement on the services and pricing. To start the contract process, you should know about different pricing structures and how they work with agile projects. After you understand these approaches, you see how to create a contract.

### ***Cost structures***

When you are procuring services for an agile project, it is important to know the difference between a *fixed-price* project, a *fixed-time* project, a *time-and-materials* project, and a *not-to-exceed* project. Each approach has its own strengths in an agile setting:

- » **Fixed-price project:** Starts out with a set budget. In a fixed-price project, a vendor works on the product and creates releases until that vendor has spent all the money in the budget or until you have delivered enough product features, whichever comes first.

For example, if you have a \$250,000 budget, and your vendor costs are \$10,000 a week, the vendor's portion of the project will be able to last 25 weeks. Within those 25 weeks, the vendor creates and releases as much shippable functionality as possible.

- » **Fixed-time project:** Has specific deadlines. For example, you may need to launch a product in time for the next holiday season, for a specific event, or to coincide with the release of another product. With fixed-time projects, you determine costs based on the cost of the vendor's team for

the duration of the project, along with any additional resource costs, such as hardware or software.

- » **Time-and-materials project:** Is more open-ended than fixed-priced or fixed-time projects. In a time-and-materials project, your work with the vendor lasts until enough product functionality is complete, without regard to total project cost. You know the total project cost at the end of the project, after your stakeholders have determined that the product has enough features to call the project complete.

For example, suppose your project costs \$10,000 a week. After 20 weeks, the stakeholders feel that they have enough valuable product features, so your project cost is \$200,000. If the stakeholders instead deem that they have enough value by the end of 10 weeks, the project cost is half that amount, \$100,000.

- » **Not-to-exceed project:** Is a project in which time and materials have a fixed-price cap.



**REMEMBER** Regardless of the cost approach, on agile projects, concentrate on completing the highest-value product features first.

## THE FALLACY OF LOW-BALLING THE VENDOR

Trying to bully vendors into providing the lowest possible price is always a lose-lose situation. Contractors in industries where projects always go to the lowest bidder have a saying: *Bid it low, and watch it grow.* It is common for vendors to provide a low price during a project's proposal process and then add multiple change orders until the buyer ends up paying as much or more than he or she would have for higher-priced offers.

Waterfall project management supports this practice by locking in scope and price at the project start, when you know almost nothing about the project. Change orders — and their accompanying cost increases — are inevitable.

A better model is for the vendor and buyer to collaborate on defining the project scope, within fixed cost and schedule constraints, as the project unfolds. Both parties can reap the benefits of what they learn during the project, and you end up with a better product full of the highest-value functionality delivered and identified at the end of each sprint. Instead of trying to be a tough negotiator, be a good collaborator.

## **Contract creation**

After you know the project's cost approach, the scrum master might help create a contract. Contracts are legally binding agreements between buyers and sellers that set expectations about work and payment.

The person responsible for creating contracts differs by organization. In some cases, a person from the legal or procurement department drafts a contract and then asks the scrum master to review it. In other cases, the opposite occurs: The scrum master drafts the contract and has a legal or procurement expert review it.

Regardless of who creates the contract, the scrum master usually acts on behalf of the scrum team to do any of the following: Initiate the contract creation, negotiate the contract details, and route the contract through any necessary internal approvals.

The agile approach of placing value on collaboration over negotiation is a key to maintaining a positive relationship between a buyer and a seller while creating and negotiating a contract. The scrum master works closely with the vendor and communicates openly and often with the vendor throughout the contract creation process.



**WARNING** The Agile Manifesto does *not* state that contracts are unnecessary.

Regardless of the size of your company or organization, it is a very good idea to create a contract between your company and your vendor for services. Skipping the contract can leave buyers and sellers open to confusion about expectations, unfinished work, and even legal problems.

At the very least, most contracts have legal language describing the parties and the work, the budget, the cost approach, and payment terms. A contract for an agile project may also include the following:

- » **A description of the work that the vendor will complete:** The vendor may have its own product vision statement, which may be a good starting point to describe the vendor's work. You may want to refer to the product vision statement in [Chapter 7](#).
- » **Agile approaches that the vendor may use:** They may include

- Meetings that the vendor will attend, such as the daily scrum, sprint planning, sprint review, and sprint retrospective meetings
- Delivery of working functionality at the end of each sprint
- The definition of done (discussed in [Chapter 9](#)): work that is developed, tested, integrated, and documented, per an agreement between the product owner, the development team, and the scrum master
- Artifacts that the vendor will provide, such as a sprint backlog with a burndown chart for status
- People whom the vendor will have on the project, such as the development team
- Where the vendor will work, such as on-site at your company
- Whether the vendor will work with its own scrum master and product owner, or if it will work with your scrum master and product owner
- A definition of what may constitute the end of the engagement: the end of a fixed budget or fixed time, or enough complete, working functionality

**» For a vendor that doesn't use an agile approach, a description of how the vendor and the vendor's work will integrate with the buyer's development team and sprints.**

This is not a comprehensive list; contract items vary by project and organization.

The contract will likely go through a few rounds of reviews and changes before the final version is complete. One way to clearly communicate changes and maintain a good relationship with a vendor is to speak with the vendor each time you propose a change. If you email a revised contract, follow up with a call to explain what you changed and why, to answer any questions, and to discuss any ideas for further revision. Open discussion helps the contract process to be positive.

If anything substantial about the vendor's services changes during contract discussions, it is a good idea for the product owner or the scrum master to review those changes with the development team. The development team

especially needs to know and provide input about changes to the service the vendor will provide, the vendor's approach, and the people on the vendor's team.



**TIP** It is quite likely that your company and the vendor will require reviews and approvals by people outside their respective project teams. People who review contracts might include high-level managers or executives, procurement specialists, accounting people, and company attorneys. This differs by organization; the scrum master needs to ensure that anyone who needs to read the contract does so.

Now that you understand a little about how to select a vendor and create a contract, it's time to look at how procurement differs among companies and organizations.

## ***Organizational considerations for procurement***

The way your company approaches procurement will make a difference in how you go about selecting a vendor and creating and negotiating a contract. Because procurement involves money and legal contracts, purchase procedures and decisions are sometimes outside a project team's control. Considerations for procurement activities can include the following:

- » **Company or organization size and experience:** Smaller and newer companies may have less formality, allowing more autonomy over purchases. Larger and more established companies tend to have more overhead with purchasing. Some companies have entire departments with people working full-time on procurement.
- » **Company or organization type:** Some organizations, such as government agencies, have legally required procurement processes and documents to complete. Private companies may have fewer restrictions on procurement than publicly traded companies because of differences in laws for public companies.
- » **Company or organization culture:** Many organizations involve the project team in procurement decisions. However, this is not always the case, and project teams sometimes find themselves working with goods or

service providers they had no part in choosing. Some companies are rather informal and don't require much documentation or process for procurement. Other companies require documents to justify the need for a good or service, formal proposals from sellers, and multiple approvals at each step in procurement.

If you're working on an agile project in an organization with heavy procurement processes and a separate procurement department, you must balance those processes with agile processes. A good way to ensure agile processes during procurement is for the scrum master to work closely with the procurement department staffers.

In [Chapter 6](#), we note that the scrum master makes sure that the organization follows agile practices and principles. In this role, the scrum master helps explain agile approaches to procurement specialists. The scrum master may find it worthwhile to help adjust organizational requirements to support agile processes.



**REMEMBER** The scrum master makes sure procurement people understand why a contract may need to accommodate changing requirements and iterations. The scrum master sets the tone for the contract creation process to be collaborative.

If an agile project team has support from an organization's upper management, it will usually be easier to work agile approaches into an organization's procurement processes.



**TIP** One good way to get support for moving agile approaches into your organization's procurement processes is to ensure that upper management understands how agile methods enable agile teams and organizations to deliver higher customer value more often. Benefits such as better product quality, reduced risk, and more control and visibility of project performance help make a strong argument for using agile processes when working with vendors. [Chapter 19](#) provides a list of key benefits of agile project management.

Organizations with light or no procurement processes provide different challenges for an agile project — or for any project, for that matter. Scrum masters may find themselves starting procurement activities from scratch, with little precedent or support.



**WARNING** People who sign contracts should have the authorization to make financial decisions for a company, and they often are people at the executive level. Scrum masters and product owners usually don't have this type of authorization. When in doubt, ask around. Find the right signatory.

After you choose a vendor and have a signed contract, the vendor can start work. In the next section, you see that, like the initial procurement processes, working with vendors has special considerations for agile projects.

## ***Working with a vendor***

How you work with a vendor on an agile project depends in part on the vendor team's structure. In an ideal situation, vendor teams are fully integrated with the buyer's organization. The vendor's team members are collocated with the buyer's scrum team. Vendor team members work as part of the buyer's development team for as long as necessary.



**TIP** Some development teams include vendor team members in their daily scrum meetings. This can be a good way to get an idea of what the vendor team is doing every day and to help the development team work more closely with the vendor. You can also invite vendors to your sprint reviews to keep them informed on your progress.

Vendor teams also can be integrated but dislocated. If the vendor can't work on-site at the buyer's company, it can still be part of the buyer's scrum team. [Chapter 14](#) has more information on team dynamics on agile projects.

If a vendor can't be collocated, or if the vendor is responsible for a discrete, separate part of the product, the vendor may have a separate scrum team. The vendor's scrum team works on the same sprint schedule as the buyer's scrum

team. See [Chapters 13](#) and [17](#) to find out how to work with more than one scrum team on a project.

If a vendor doesn't use agile project management processes, the vendor's team works separately from the buyer's scrum team, outside the sprints, and on its own schedule. The vendor's traditional project manager helps ensure that the vendor can deliver its services when the development team needs them. The buyer's scrum master may need to step in if the vendor's processes or timeline becomes a roadblock or a disruption for the development team. See the “[Managing projects with dislocated teams](#)” section in [Chapter 14](#) for information about working with non-agile teams.

Vendors may provide services for a defined amount of time, or for the life of the project. After the vendor's work is complete, the contract is closed.

## ***Closing a contract***

After a vendor completes work on a contract, the buyer's scrum master usually has some final tasks to close the contract.

If the project finishes normally, according to the contract terms, the scrum master may want to acknowledge the end of the contract in writing. If the project is a time-and-materials project, the scrum master should definitely end it in writing to ensure that the vendor doesn't keep working on lower-priority requirements — and billing for them.

Depending on the organizational structure and the contract's cost structure, the scrum master may be responsible for notifying the buyer's company accounting department after work is complete to ensure that the vendor is paid properly.

If the project finishes before the contract dictates the end, the scrum master needs to notify the vendor in writing and follow any early termination instructions from the contract.



**TIP** End the engagement on a positive note. If the vendor did a good job, the scrum team may want to acknowledge the people on the vendor's team at the sprint reviews. Everyone on the project could potentially work together again, and a simple, sincere “thank-you” can help

maintain a good relationship for future projects.

# Chapter 13

# Managing Time and Cost

---

## IN THIS CHAPTER

- » Understanding what's unique about time management on agile projects
- » Finding out how to manage time on agile projects
- » Recognizing how cost management is different on agile projects
- » Seeing how to manage cost on agile projects

Managing project time and controlling project costs are key aspects of managing a project. In this chapter, you see agile approaches to time and cost management. You find out how to use a scrum team's development speed to determine time and cost on a given project and how to increase development speed to lower your project's time and cost.

## *What's Different about Agile Time Management?*

In project management terms, *time* refers to the processes that ensure timely project completion. To understand agile time management, it helps to review some of the Agile Principles we discuss in [Chapter 2](#):

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
8. Agile processes promote sustainable development. The sponsors,

developers, and users should be able to maintain a constant pace indefinitely.

[Table 13-1](#) shows some of the differences between time management on traditional projects and on agile projects.

## TABLE 13-1 Traditional versus Agile Time Management

<i>Time Management with Traditional Approaches</i>	<i>Time Management with Agile Approaches</i>
Fixed scope directly drives the schedule.	Scope is not fixed on agile projects. Time can be fixed, and development teams can create the requirements that will fit into a specific time frame.
Project managers determine time based on the requirements gathered at the beginning of the project.	During the project, scrum teams assess and reassess how much work they can complete in a given time frame.
Teams work at one time in phases on all project requirements, such as requirements gathering, design, development, testing, and deployment. No schedule difference exists between critical requirements and optional requirements.	Scrum teams work in sprints and complete all the work on the highest-priority, highest-value requirements first.
Teams do not start actual product development until later in the project, after the requirements-gathering and design phases are complete.	Scrum teams start product development in the first sprint.
Time is more variable on traditional projects.	Timeboxed sprints on agile projects stay stable, enabling predictability.
Project managers try to predict schedules at the project start, when they know little about the product.	Scrum teams determine long-range schedules on actual development performance in sprints. Scrum teams adjust time estimates throughout the project as they learn more about the product and the development team's speed, or <i>velocity</i> . You find more about velocity later in this chapter.



**REMEMBER** Fixed-schedule and fixed-price projects have lower risk with agile techniques because agile development teams always deliver the highest-priority functionality within the time or budget constraints.

A big benefit of agile time management techniques is that agile project teams can deliver products much earlier than traditional project teams. For example, starting development earlier and completing functionality in iterations often allow agile project teams that work with our company, Platinum Edge, to bring value to the market 30 percent to 40 percent faster.



TIP The reason agile projects finish sooner isn't complicated; they simply start development sooner.

In the next section, find out how to manage time on an agile project.

## *Managing Agile Schedules*

Agile practices support both strategic and tactical schedules and time management:

- » Your early planning is strategic in nature. The high-level requirements in the product roadmap and the product backlog can help you get an early idea of the overall schedule. Find out how to create a product roadmap and product backlog in [Chapter 7](#).
- » Your detailed planning for each release and at each sprint is tactical. Read more about release planning and sprint planning in [Chapter 8](#).
  - At release planning, you can plan your release to match a specific date, with minimal marketable features.
  - You also can plan your release with enough time to create a specific set of features.
  - During each sprint planning meeting, in addition to selecting the scope for the sprint, the development team estimates the time, in hours, to complete individual tasks for each of that sprint's requirements. Use the sprint backlog to manage detailed time allocations throughout the sprint.
- » After your project is underway, use the scrum team's velocity (development speed) to fine-tune your scheduling. We discuss velocity in the next section.

### **DETERMINING AN AGILE PROJECT'S LENGTH**

A few factors determine the length of agile projects:

- **Assigned deadline:** For business reasons, agile project teams may want to set a specific end date. For example, you may want to get a product to market for a specific shopping season or to coincide with the timing of a competitor's product release. In that case, you set a specific end date, and create as much shippable functionality as possible from the project start until the end date.
- **Budget considerations:** Agile project teams may also have budget considerations that affect the amount of time a project will last. For example, if you have a \$1,600,000 budget, and your project costs \$20,000 a week to run, your project will be able to last 80 weeks. You'll have 80 weeks to create and release as much shippable functionality as possible.
- **Functionality completed:** Agile projects may also last only until enough functionality is complete. Project teams may run sprints until the requirements with the highest value are complete, and then determine that the lower-value requirements — the ones that few people will use or that will not generate much revenue — aren't necessary.



**REMEMBER** In [Chapter 8](#), we describe planning releases for *minimal marketable features*, the smallest group of product functionality providing enough value that you can effectively deploy and promote in the marketplace.

To determine how much functionality an agile development team can deliver within a set amount of time, you need to know your development team's velocity. In the next section, you take a look at how to measure velocity, how to use velocity for a project timeline, and how to increase velocity throughout the project.

## *Introducing velocity*

One of the most important things about time management on agile projects is the use of velocity, a powerful tool for forecasting long-term timelines. *Velocity*, in agile terms, is a development team's work speed. In [Chapter 7](#), we describe measuring the effort for requirements, or user stories, in story points. You measure velocity by the number of user story points that the development team completes in each sprint.



**REMEMBER** A *user story* is a simple description of a product requirement, identifying what a requirement must accomplish, and for whom. User

story points are relative numbers that describe the amount of effort necessary to develop a user story. [Chapter 8](#) delves into the details of creating user stories and estimating the effort using story points.

When you know the development team's velocity, you can use it as a long-range planning tool. Velocity can help you forecast how long the scrum team will take to complete a certain number of requirements and how much a project may cost.

In the next section, you dive into velocity as a tool for time management. You see how scope changes affect an agile project's timeline. You also find out how to work with multiple scrum teams and review agile artifacts for time management.

## ***Monitoring and adjusting velocity***

After a project starts, the scrum team starts to monitor its velocity. You measure velocity from sprint to sprint. You use velocity for long-term schedule and budget planning as well as for sprint planning.

In general, people are good at planning and estimating in the short term, so identifying hours for tasks in an upcoming sprint works well. At the same time, people are often terrible at estimating distant tasks in absolute terms such as hours. Tools such as relative estimating and velocity, which are based on performance, are more accurate measurements for longer-term planning.

Velocity is a good trending tool. You can use it to determine future timelines because the activities and development time within sprints is the same from sprint to sprint.



**WARNING** Velocity is a post-sprint fact, not a goal. Avoid attempting to guess or commit to a certain velocity before a project starts or in the middle of a sprint. You'll only set unrealistic expectations about how much work the team can complete. If velocity turns into a target rather than a past measurement, scrum teams may be tempted to exaggerate estimated story points to meet that target, rendering velocity meaningless. Instead, use the scrum team's actual velocity to forecast how much longer the project may take and cost. Also focus on increasing velocity by removing constraints identified during the sprint and at the sprint

retrospective.

In the next section, you see how to calculate velocity, how to use velocity to predict a project's schedule, and how to increase your scrum team's velocity.

## ***Calculating velocity***

At the end of each sprint, the scrum team looks at the requirements it has finished and adds up the number of story points associated with those requirements. The total number of completed story points is the scrum team's velocity for that sprint. After the first few sprints, you will start to see a trend and will be able to calculate the average velocity.



**WARNING** Because velocity is a number, managers and executives may be tempted to use velocity as a performance metric for compensating and comparing teams. Velocity is not a performance metric, is team-specific, and should not be used outside the scrum team. It is no more than a planning tool scrum teams can use to forecast remaining work.

The *average velocity* is the total number of story points completed, divided by the total number of sprints completed. For example, if the development team's velocity was

Sprint 1 = 15 points

Sprint 2 = 13 points

Sprint 3 = 16 points

Sprint 4 = 20 points

your total number of story points completed will be 64. Your average velocity will be 16: 64 story points divided by 4 sprints.

After you have run a sprint and know the scrum team's velocity, you can start forecasting the remaining time on your project.

## ***Using velocity to estimate the project timeline***

When you know your velocity, you can determine how long your project will last. Follow these steps:

- 1. Add up the number of story points for the remaining requirements in the product backlog.**
- 2. Determine the number of sprints you'll need by dividing the number of story points remaining in the product backlog by the velocity:**
  - To get a pessimistic estimate, use the lowest velocity the development team has accomplished.
  - To get an optimistic estimate, use the highest velocity the development team has accomplished.
  - To get a most likely estimate, use the average velocity the development team has accomplished.



TIP Using this empirical data — actual output speed — a product owner can give stakeholders a range of release outcomes, and they can work together to make business prioritization decisions early in the project. These decisions might include whether there is a need to spin up an additional scrum team to develop more scope items, adjust market release dates, or request project budget.

- 3. Determine how much time it will take to complete the story points in the product backlog by multiplying sprint length by the number of remaining sprints.**

For example, assume that

- Your remaining product backlog contains 800 story points.
- Your development team velocity averages 20 story points per sprint.

How many more sprints will your product backlog need? Divide the number of story points by your velocity, and you get your remaining sprints. In this case,  $800/20 = 40$ .

If you're using two-week sprints on your project, your project will last 80 weeks.

After the scrum team knows its velocity and the number of story points for the requirements, you can use the velocity to determine how long any given group of requirements will take to create. For example:

- » You can calculate the time an individual release may take if you have an idea of the number of story points that will go into that release. At the release level, your story point estimates will be more high level than at the sprint level. If you're basing your release timing on delivering specific functionality, your release date may change as you refine your user stories and estimates throughout the project.
- » You can calculate the time you need for a specific group of user stories — such as all high-priority stories or all stories relating to a particular theme — by using the number of story points in that group of user stories.

Velocity differs from sprint to sprint. In the first few sprints, when the project is new, the scrum team will typically have a low velocity. As the project progresses, velocity should increase because the scrum team will have learned more about the product and will have matured as a team working together. Setbacks within specific sprints can temporarily decrease velocity from time to time, but agile processes such as the sprint retrospective can help the scrum team ensure that those setbacks are temporary.



**TIP** In the beginning of a project, velocity will vary considerably from sprint to sprint. Velocity will become more consistent over time, as long as the scrum team members remain consistent.

Scrum teams can also increase their velocity throughout agile projects, making projects shorter and less costly. In the next section, you find ways to increase velocity in each consecutive sprint.

### ***Increasing velocity***

If a scrum team has a product backlog with 800 story points and an average velocity of 20 story points, the project will last 40 sprints — 80 weeks, with 2-week sprints. But what if the scrum team could increase its velocity?

- » Increasing the average velocity to 23 story points per sprint would mean 34.78 sprints. If you round that up to 35 sprints, the same project would last 70 weeks.
- » An average velocity of 26 would take about 31 sprints, or 62 weeks.

- » An average velocity of 31 would take about 26 sprints, or 52 weeks.

As you can see, increasing velocity can save a good deal of time and, consequently, money.

Velocity can naturally increase with each sprint, as the scrum team finds its rhythm of working together on the project. However, opportunities exist to also raise velocity on agile projects, past the common increases that come with time. Everyone on a scrum team plays a part in helping get higher velocity with every successive sprint:

- » **Remove project roadblocks:** One way to increase velocity is to quickly remove project roadblocks, or impediments. Roadblocks are anything that keeps a development team member from working to full capacity. By definition, roadblocks can decrease velocity. Clearing roadblocks as soon as they arise increases velocity by helping the scrum team to be fully functional and productive. Find out more about removing project impediments in [Chapter 9](#).
- » **Avoid project roadblocks:** The best way to increase velocity is to strategically create ways to avoid roadblocks in the first place. By knowing — or learning about — the processes and the specific needs of groups your team will work with, you can head off roadblocks before they arise.
- » **Eliminate distractions:** Another way to increase velocity is for the scrum master to protect the development team from distractions. By making sure people don't request work outside the sprint goal from the development team — even tasks that might take a small amount of time — the scrum master will be able to help keep the development team focused on the sprint.



REMEMBER Having a dedicated scrum master who continually helps remove constraints for the scrum team will result in continually increasing velocity. The value of a dedicated scrum master is quantifiable.

- » **Solicit input from the team:** Finally, everyone on the scrum team can provide ideas for increasing velocity in the sprint retrospective meeting.

The development team knows its work the best, and may have ideas on how to improve output. The product owner may have insights into the requirements that can help the development team work faster. The scrum master will have seen any repetitive roadblocks and can discuss how to prevent the roadblocks in the first place.

## PREVENTING ROADBLOCKS

One development team we worked with needed feedback from its company's legal department but had not been able to get a response via email or voicemail. In a daily scrum meeting, one of the development team members stated this lack of response as a roadblock. After the scrum meeting was over, the scrum master walked over to the legal department and found the right person to work with. After talking to that person, the scrum master found out that her email was constantly flooded with requests, and her voicemail was not much better.

The scrum master then suggested a process for future legal requests: Moving forward, the development team members could walk over to the legal department with requests and get feedback right there, in person, immediately. The new process took only a few minutes, but saved days on turnaround from the legal department, effectively preventing similar roadblocks in the future. Finding ways to prevent roadblocks helps increase the scrum team's velocity.



**TIP** Increasing velocity is valuable, but remember that you may not see changes overnight. Scrum team velocity often has a pattern of slow increases, some big velocity jumps, a flat period, and then slow increases again as the scrum team identifies, experiments, and corrects constraints that are holding it back.

### *Consistency for useful velocity*

Because velocity is a measure of work completed in terms of story points, it's an accurate indicator and predictor of project performance only when you use the following practices:

- » **Consistent sprint lengths:** Each sprint should last the same amount of time throughout the life of the project. If sprint lengths are different, the amount of work the development team can complete in each sprint will be different, and velocity won't be relevant in predicting the remaining time on the project.
- » **Consistent work hours:** Individual development team members should

work the same number of hours in each sprint. If Sandy works 45 hours in one sprint, 23 in another, and 68 in yet another, she will naturally complete a different amount of work from sprint to sprint. However, if Sandy always works the same number hours in one sprint, her velocity will be comparable between sprints.

- » **Consistent development team members:** Different people work at different rates. Tom might work faster than Bob, so if Tom works on one sprint and Bob works on the next sprint, the velocity of Tom's sprint will not be a good prediction for Bob's sprint.

When sprint lengths, work hours, and team members remain consistent throughout a project, you can use velocity to truly know whether development speed is increasing or decreasing and to accurately estimate the project timeline.



**WARNING** Performance does not scale linearly with available time. For example, if you have two-week sprints with 20 story points per sprint, going to three-week sprints does not guarantee 30 story points. The new sprint length will generate an unknown change in velocity.

Although changing sprint lengths does introduce variance into a scrum team's velocity and projections, we rarely discourage scrum teams from decreasing their sprint lengths (from three weeks to two, or from two weeks to one) because shorter feedback loops allow scrum teams to react faster to customer feedback, enabling them to deliver more value to their customer. However, changing sprint lengths always comes with the same caution: Velocity does not scale linearly in the opposite direction either, and scrum teams will have to establish a new velocity for their shorter sprint before their projections will become reliable again.

When you know how to accurately measure and increase velocity, you have a powerful tool for managing time and cost on a project. In the next section, we talk about how to manage a timeline in an ever-changing agile environment.

## ***Managing scope changes from a time perspective***

Agile project teams welcome changing requirements at any time throughout a

project, which means project scope reflects the real priorities of the business. It is “requirements Darwinism” at its purest — development teams complete requirements of highest priority first. Fixed sprint lengths force out requirements that sound like good ideas in theory but never win the “either this requirement or that requirement” contest.

New requirements may have no effect on a project’s timeline; you just have to prioritize. Working with the project stakeholders, the product owner can determine to develop only the requirements that will fit in a certain window of time or budget. The priority ranking of items in the product backlog determines which requirements are important enough to develop. The scrum team can guarantee completing higher-priority requirements. The lower-priority requirements might be part of another project or may never be created.



**REMEMBER** In [Chapter 12](#), we discuss how to manage scope changes with the product backlog. When you add a new requirement to an agile project, you prioritize that requirement against all other items in your product backlog and add the new item into the appropriate spot in the product backlog. This may move other product backlog items down in priority. If you keep your product backlog and its estimates up-to-date as new requirements arise, you’ll always have a good idea of the project timeline, even with constantly changing scope.

On the other hand, the product owner and the project stakeholders may determine that all the requirements in the product backlog, including new requirements, are useful enough to include in the project. In this case, you extend the project end date to accommodate the additional scope, increase velocity, or divide the project scope among multiple scrum teams that will work simultaneously on different product features. Learn more about multi-team projects in [Chapter 17](#).

Project teams often make schedule decisions about lower-priority requirements toward the end of a project. The reasons for these just-in-time decisions are because marketplace demands for specific scope items change, and also because velocity tends to increase as the development team gets into a rhythm. Changes in velocity increase your predictions about how many

product backlog items the development team can complete in a given amount of time. On agile projects, you wait until the last responsible moment — when you know the most about the question at hand — to make decisions you'll be committed to for the rest of the project.

The next section shows you how to work with more than one scrum team on a project.

## ***Managing time by using multiple teams***

For larger projects, multiple scrum teams working in parallel may be able to complete a project in a shorter time frame.

You may want to create a project with multiple scrum teams if

- » Your project is very large and will require more than a single development team of nine or fewer development team members to complete.
- » Your project has a specific end date that you must meet, and the scrum team's velocity will not be sufficient to complete the most valuable requirements by that end date.



**REMEMBER** The ideal size for a development team on an agile project is no less than three and no more than nine people. Groups of more than nine people start to build silos, and the number of communication channels makes self-management more difficult. (In some cases, we've seen these issues in teams smaller than nine.) When your product development requires more development team members than can effectively communicate, it may be time to consider using multiple scrum teams.

If you have multiple scrum teams on a project, break the work into themes, or logical groups of product features, for each team.

Before rushing into that, though, you need to consider the overall scope of the themes and the relationship between them. The work needs to be sufficiently separate to allow the teams to operate independently, with as few interdependencies as possible. In [Chapter 17](#), we show you several techniques for scaling product development work across multiple teams.

## ***Using agile artifacts for time management***

The product roadmap, product backlog, release plan, and sprint backlog all play a part in time management. [Table 13-2](#) shows how each artifact contributes to time management.

**[TABLE 13-2 Agile Artifacts and Time Management](#)**

<b><i>Artifact</i></b>	<b><i>Role in Time Management</i></b>
<b>Product roadmap:</b> The product roadmap is a prioritized, holistic view of the high-level requirements that support the product's vision. Find more about the product roadmap in <a href="#">Chapter 7</a> .	The product roadmap is a strategic look at the overall project priorities. Although the product roadmap likely will not have specific dates, it will have general date ranges for groups of functionality and will allow an initial framing for bringing the product to market.
<b>Product backlog:</b> The product backlog is a complete list of all currently known product requirements. Find more about the product backlog in <a href="#">Chapters 7 and 8</a> .	The requirements in your product backlog will have estimated story points. After you know your development team's velocity, you can use the total number of story points in the product backlog to determine a realistic project end date.
<b>Release plan:</b> The release plan contains a release schedule for a minimum set of requirements. Find more about the release plan in <a href="#">Chapter 8</a> .	The release plan will have a target release date for a specific goal supported by a minimal set of marketable functionalities. Scrum teams plan and work on only one release at a time.
<b>Sprint backlog:</b> The sprint backlog contains the requirements and tasks for the current sprint. Find more about the sprint backlog in <a href="#">Chapter 8</a> .	During your sprint-planning meeting, you estimate individual tasks in the backlog in hours. At the end of each sprint, you take the total completed story points from the sprint backlog to calculate your development team's velocity for that sprint.

In the next sections, you dive into cost management for agile projects. Cost management is directly related to time management. You compare traditional approaches to cost management to those in agile projects. You find out how to estimate costs on an agile project and how to use velocity to forecast your long-term budget.

## ***What's Different about Agile Cost Management?***

*Cost* is a project's financial budget. When you work on an agile project, you focus on value, exploit the power of change, and aim for simplicity. Agile Principles 1, 2, and 10 state the following:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
10. Simplicity — the art of maximizing the amount of work not done — is essential.

Because of this emphasis on value, change, and simplicity, agile projects have a different approach to budget and cost management than traditional projects. [Table 13-3](#) highlights some of the differences.

**TABLE 13-3 Traditional versus Agile Cost Management**

<i>Cost Management with Traditional Approaches</i>	<i>Cost Management with Agile Approaches</i>
Cost, like time, is based on fixed scope.	Project schedule, not scope, has the biggest effect on cost. You can start with a fixed cost and a fixed amount of time, and then complete requirements as potentially shippable functionality that fit into your budget and schedule.
Organizations estimate project costs and fund projects before the project starts.	Product owners often secure project funding after the product roadmap stage is complete. Some organizations even fund agile projects one release at a time; product owners will secure funding after completing release planning for each release.
New requirements mean higher costs. Because project managers estimate costs based on what they know at the project start, which is very little, cost overruns are common.	Project teams can replace lower-priority requirements with new, equivalently sized high-priority requirements with no effect on time or cost.
Scope bloat (see <a href="#">Chapter 12</a> ) wastes large amounts of money on features that people simply do not use.	Because agile development teams complete requirements by priority, they concentrate on creating only the product features that users need, whether those features are added on day 1 or day 100 of the project.
Projects cannot generate revenue until the project is complete.	Project teams can release working, revenue-generating functionality early, creating a self-funding project.



TECHNICAL STUFF

When costs increase, project sponsors sometimes find themselves in a kind of hostage situation. A waterfall approach does not call for any complete product functionality until the end of a project. Because traditional approaches to development are all-or-nothing proposals, if costs increase and stakeholders don't pay more for the product, they will not get *any* finished requirements. The incomplete product becomes a

kidnapped hostage; pay more, or get nothing.

In the following sections, you find out about cost approaches in agile projects, how to estimate costs for an agile project, how to control your budget, and how to lower costs.

## ***Managing Agile Budgets***

On agile projects, cost is mostly a direct expression of project time. Because scrum teams consist of full-time, dedicated team members, they have a set team cost — generally expressed as an hourly or fixed rate per person — that should be the same for each sprint. Consistent sprint lengths, work hours, and team members enable you to accurately use velocity to predict development speed. Once you use velocity to determine how many sprints your project will take — that is, how long your project will be — you can know how much your scrum team will cost for the whole project.

Project cost also includes the cost for resources like hardware, software, licenses, and any other supplies you might need to complete your project.

In this section, you find out how to create an initial budget and how to use the scrum team's velocity to determine long-range costs.

### ***Creating an initial budget***

To create your project budget, you need to know the cost for your scrum team, per sprint, and the cost for any additional resources you need to complete the project.

Typically, you calculate the cost for your scrum team by using an hourly rate for each team member. Multiply each team member's hourly rate by his or her available hours per week by the number of weeks in your sprints to calculate your scrum team's per-sprint cost. [Table 13-4](#) shows a sample budget for a scrum team — the product owner, five development team members, and the scrum master — for a two-week sprint.

**TABLE 13-4** Sample Scrum Team Budget for a Two-Week Sprint

<i>Team Member</i>	<i>Hourly Rate</i>	<i>Weekly Hours</i>	<i>Weekly Cost</i>	<i>Sprint Cost (2 Weeks)</i>
Don	\$80	40	\$3,200	\$6,400

Peggy	\$70	40	\$2,800	\$5,600
Bob	\$70	40	\$2,800	\$5,600
Mike	\$65	40	\$2,600	\$5,200
Joan	\$85	40	\$3,400	\$6,800
Tommy	\$75	40	\$3,000	\$6,000
Pete	\$55	40	\$2,200	\$4,400
Total		280	\$20,000	\$40,000

The cost for additional resources will vary by project. In addition to scrum team member costs, take the following into account when determining your project costs:

- » Hardware costs
- » Software, including license costs
- » Hosting costs
- » Training costs
- » Miscellaneous team expenses, such as additional office supplies, team lunches, travel costs, and the price of any tools you may need

These costs may be one-time costs, rather than per-sprint costs. We suggest separating these costs in your budget; as you see in the next section, you need your cost for each sprint to determine the cost for the project. (To keep calculations simple throughout this chapter, we assume that the project cost of \$40,000 includes scrum team member costs as well as any additional resources, such as those just listed.)



TIP *Resources* typically refer to inanimate objects, not people. Resources need to be managed. When discussing resources on a project, refer to people as *team members*, *talent*, or just *people*. This issue may seem minor, but the more you focus on individuals and interactions over processes and tools, even in the details, the more your mindset will change to think and be more agile.

## ***Creating a self-funding project***

A big benefit of agile projects is the capability to have a self-funding project. Scrum teams deliver working functionality at the end of each sprint and make that functionality available to the marketplace at the end of each release cycle. If your product is an income-generating product, you could use revenue from early releases to help fund the rest of your project.

For example, an ecommerce website might generate \$15,000 a month in sales after the first release, \$40,000 a month after the second release, and so on. [Tables 13-5](#) and [13-6](#) compare income on a sample traditional project to the income from a self-funding agile project.

**TABLE 13-5 Income from a Traditional Project with a Final Release after Six Months**

<i>Month</i>	<i>Income Generated</i>	<i>Total Project Income</i>
January	\$0	\$0
February	\$0	\$0
March	\$0	\$0
April	\$0	\$0
May	\$0	\$0
June	\$0	\$0
July	\$100,000	\$100,000

**TABLE 13-6 Income from a Project with Monthly Releases and a Final Release after Six Months**

<i>Month/Release</i>	<i>Income Generated</i>	<i>Total Project Income</i>
January	\$0	\$0
February	\$15,000	\$15,000
March	\$25,000	\$40,000
April	\$40,000	\$80,000
May	\$70,000	\$150,000
June	\$80,000	\$230,000
July	\$100,000	\$330,000

In [Table 13-5](#), the project created \$100,000 in income after six months of development. Now compare the income in [Table 13-5](#) to the income generated in [Table 13-6](#).

In [Table 13-6](#), the project generated income with the first release. By the end of six months, the project had generated \$330,000 — \$230,000 more than the project in [Table 13-5](#).

## ***Using velocity to determine long-range costs***

The “[Using velocity to estimate the project timeline](#)” section, earlier in this chapter, shows you how to determine how much time a project will take, using the scrum team’s velocity and the remaining story points in the product backlog. You can use the same information to determine the cost for the project or for your current release.

After you know the scrum team’s velocity, you can calculate the cost for the remainder of the project.

In the velocity example from earlier in this chapter, where your scrum team velocity averages 16 story points per sprint, your product backlog contains 800 story points, and your sprints are 2 weeks long, your project will take 50 sprints, or 100 weeks, to complete.

To determine the remaining cost for your project, multiply the cost per sprint by the number of sprints the scrum team needs to complete the product backlog.

If your scrum team cost is \$40,000 per sprint and you have 50 sprints left, your remaining cost for your project will be \$2,000,000.

In the next sections, you find out different ways to lower your project costs.

### ***Lowering cost by increasing velocity***

In the time management section of this chapter, we talk about increasing the scrum team’s velocity. Using the examples from the earlier section, and the \$40,000 per two-week sprint from [Table 13-4](#), increasing velocity could reduce your costs, as follows:

- » If the scrum team increases its average velocity from 16 to 20 story points per sprint
  - You will have 40 remaining sprints.

- Your project will cost \$1.6 million, saving you more than \$400,000.
- » If the scrum team increases its velocity to 23 story points
- You will have 35 remaining sprints.
  - Your project will cost \$1.4 million, saving you an additional \$200,000.
- » If the scrum team increases its velocity to 26 story points
- You will have 31 remaining sprints.
  - Your project will cost \$1.24 million, an additional \$160,000 savings.

As you can see, increasing the scrum team's velocity by removing impediments can provide real savings on project costs. See how to help the scrum team become more productive in the "[Increasing velocity](#)" section, earlier in this chapter.

### ***Lowering cost by reducing time***

You can also lower your project costs by not completing lower-priority requirements, thus lowering the number of sprints you need. Because completed functionality is delivered with each sprint in an agile project, the project stakeholders can make a business decision to end a project when the cost of future development is higher than the value of that future development.

Project stakeholders can then use the remaining budget from the old project to start a new, more valuable project. The practice of moving the budget from one project to another is called *capital redeployment*.

To determine a project's end based on cost, you need to know

- » The business value (V) of the remaining requirements in the product backlog
- » The actual cost (AC) of the work it will take to complete the requirements in the product backlog
- » The opportunity cost (OC), or the value of having the scrum team work on a new project

When  $V < AC + OC$ , the project can stop because the cost you'll sink into the project will be more than the value you will receive from the project.

Consider this example: A company is running an agile project and

- » The remaining features in the product backlog will generate \$100,000 in income ( $V = \$100,000$ ).
- » It will take three sprints with a cost of \$40,000 per sprint to create those features, a total of \$120,000 ( $AC = \$120,000$ ).
- » The scrum team could be working on a new project that would generate \$150,000 after three sprints, minus the scrum team's cost ( $OC = \$150,000$ ).
- » The project value, \$100,000, is less than the actual costs plus opportunity costs, or \$270,000. This would be a good time to end the project.

The opportunity for capital redeployment sometimes arises in emergencies, when an organization needs members of the scrum team to pause a project for critical unplanned work. Project sponsors sometimes evaluate a project's remaining value and cost before restarting a paused project.



**WARNING** Pausing a project can be expensive. The costs associated with demobilization and remobilization — saving work in progress, documenting current state, debriefing paused project team members, retooling for the new project, briefing team members on the new project, learning new skills required on the new project — can be significant and should be evaluated before making the decision to pause a project that may need to be remobilized again in the future.  $V < AC + OC$  can help with this decision.

Project sponsors may also compare the product backlog value to remaining development costs throughout the project, so they know just the right time to end the project and receive the most value.

### ***Determining other costs***

Similar to time management, after you know the scrum team's velocity, you can determine the cost of anything in the project. For example:

- » You can calculate the cost for an individual release if you have an idea of the number of story points that will go into that release. Divide the number of story points in the release by the scrum team's velocity to determine how many sprints will be required. At the release, your story point estimates will be more high-level than at the sprint, so your costs may change, depending on how you determine your release date.
- » You can calculate the cost for a specific group of user stories, such as all high-priority stories or all stories relating to a particular theme, by using the number of story points in that group of user stories.

## ***Using agile artifacts for cost management***

You can use the product roadmap, release plan, and sprint backlog for cost management. [Table 13-2](#) shows how each artifact helps you measure and evaluate project time and costs.

Time and cost forecasts based on actual development team performance are more accurate than forecasts based on hope.

# Chapter 14

# Managing Team Dynamics and Communication

---

## IN THIS CHAPTER

- » Recognizing what makes agile team dynamics different
- » Finding out how to work with agile teams
- » Understanding how communication differs on agile projects
- » Seeing how communication works on agile projects

Team dynamics and communication are significant parts of project management. In this chapter, you find out about traditional and agile approaches to project teams and communication. You see how a high value on individuals and interactions makes agile project teams great teams to work on. You also find out how face-to-face communication helps make agile projects successful.

## *What's Different about Agile Team Dynamics?*

What makes a project team on an agile project unique? The core reason agile teams are different from traditional teams is their team dynamics. The Agile Manifesto (refer to [Chapter 2](#)) sets the framework for how agile project team members work together: The very first item of value in the manifesto is *individuals and interactions* over processes and tools.

The following agile principles, also from [Chapter 2](#), support valuing people on the project team and how they work together:

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The agile principles apply to many different project management areas. You see some of these principles repeated in different chapters of this book.



**REMEMBER** On agile projects, the development team contains the people who do the physical work of creating the product. The scrum team contains the development team, plus the product owner and the scrum master. The project team is the scrum team and your project stakeholder. Everyone on the scrum team has responsibilities related to self-management.

[Table 14-1](#) shows some differences between team management on traditional projects and on agile projects.

## **TABLE 14-1 Traditional versus Agile Team Dynamics**

<i>Team Management with Traditional Approaches</i>	<i>Team Dynamics with Agile Approaches</i>
Project teams rely on <i>command and control</i> — a top-down approach to project management, where the project manager is responsible for assigning tasks to team members and attempting to control what the team does.	Agile teams are self-managing, self-organizing, and benefit from <i>servant leadership</i> . Instead of top-down management, a servant-leader coaches, removes obstacles, and prevents distractions to enable the team to thrive.
Companies evaluate individual employee performance.	Agile organizations evaluate agile team performance. Agile teams, like any sports team, succeed or fail as a whole team. Whole-team performance encourages individual team members to increase the ways they can contribute to the team's success.
Team members often find themselves working on more than one project at a time, switching their attention back and forth.	Development teams are dedicated to one project at a time, and reap the benefits of focus.

Agile organizations focus on skills instead of titles.

Development team members have distinct roles, such as programmer or tester.	<small>Agile organizations focus on skills instead of titles.</small> Development teams work cross-functionally, doing different jobs within the team to ensure that they complete priority requirements quickly.
Development teams have no specific size limits.	Development teams are intentionally limited in size. Ideally, development teams have no fewer than three and no more than nine people.
Team members are commonly referred to as <i>resources</i> , a shortened term for <i>human resources</i> .	Team members are called <i>people</i> , <i>talent</i> , or simply <i>team members</i> . On an agile project, you probably will not hear the term <i>resource</i> used to refer to people.



TIP We avoid the term *resources* when referring people. Referring to people and equipment with the same term is the beginning of thinking of team members as interchangeable objects that can be swapped in and out. Resources are things, utilitarian and expendable. The people on your project team are human beings, with emotions, ideas, and priorities inside and outside the project. People can learn and create and grow throughout the project. Respecting your fellow project team members by referring to them as *people* instead of *resources* is a subtle but powerful way to reinforce the fact that people are at the core of an agile mindset.

The following sections discuss how working with a dedicated, cross-functional, self-organizing, size-limited team benefits agile projects. You find out more about servant leadership and creating a good environment for a scrum team. In short, you find out how team dynamics help agile projects succeed.

## Managing Agile Team Dynamics

Time and again, when we talk with product owners, developers, and scrum masters, we hear the same thing: People enjoy working on agile projects. Agile team dynamics enable people to do great work in the best way they know how. People on scrum teams have opportunities to learn, to teach, to lead, and to be part of a cohesive, self-managing team.

The following sections show you how to work as part of an agile team (using scrum as the context) and why agile approaches to teamwork make agile projects successful.

## **Becoming self-managing and self-organizing**

On agile projects, scrum teams are directly accountable for creating deliverables. Scrum teams manage themselves, organizing their own work and tasks. No one person tells the scrum team what to do. This doesn't mean that agile projects have no leadership. Each member of the scrum team has the opportunity to lead informally, based on his or her skills, ideas, and initiative.



**REMEMBER** On agile projects, the development team contains the people who are doing the physical work of creating the product. The scrum team contains the development team, plus the product owner and the scrum master. The project team is the scrum team and your project stakeholders. Both the development team and the overall scrum team have responsibilities related to self-management.

The idea of self-management and self-organization is a mature way of thinking about work. Self-management assumes that people are professional, motivated, and dedicated enough to commit to a job and see it through. At the core of self-management is the idea that the people who are doing a job from day to day know the most about that job and are best qualified to determine how to complete it. Working with a self-managing scrum team requires trust and respect within the team and within the team's organization as a whole.

Nonetheless, let's be clear: Accountability is at the core of agile projects. The difference is that in an agile project, teams are held accountable for tangible results that you can see and demonstrate. Traditionally, companies held teams accountable for compliance to the organization's step-by-step process — stripping them of the ability or incentive to be innovative. Self-management, however, returns innovation and creativity to development teams.



**TIP** For a scrum team to be self-managing, you need an environment of trust. Everyone on the scrum team must trust one another to do his or her best for the scrum team and the project. The scrum team's company or organization must also trust the scrum team to be competent, to make

decisions, and to manage itself. To create and maintain an environment of trust, each member of the scrum team must commit, individually and as a team, to the project and to one another.

Self-managing development teams create better product architectures, requirements, and design for a simple reason: ownership. When you give people the freedom and responsibility to solve problems, they are more mentally engaged in their work.

Scrum team members play roles in all areas of project management. [Table 14-2](#) shows how scrum teams and development teams manage scope, procurement, time, cost, team dynamics, communication, stakeholders, quality, and risk.

**TABLE 14-2 Project Management and Self-Managing Teams**

<b>Area of Project Management</b>	<b>How Product Owners Self-Manage</b>	<b>How Development Teams Self-Manage</b>	<b>How Scrum Masters Self-Manage</b>
Scope	<p>Use the product vision, the release goal, and each sprint goal to determine if and where scope items belong.</p> <p>Use product backlog prioritization to determine which requirements are developed.</p>	<p>May suggest features based on technical affinity.</p> <p>Work directly with the product owner to clarify requirements.</p> <p>Identify how much work they can take on in a sprint.</p> <p>Identify the tasks to complete scope in the sprint backlog.</p> <p>Determine the best way to create specific features.</p>	<p>Remove impediments that limit the amount of scope the development team can create.</p> <p>Through coaching, help development teams become more productive with each successive sprint.</p>
Procurement	Secure necessary funding for tools and equipment for development teams.	<p>Identify the tools they need to create the product.</p> <p>Work with the product owner to get those tools.</p>	Help procure tools and equipment that accelerate development team velocity.
Time	Ensure that the development team correctly understands product features so that development teams can correctly estimate the effort to create those features.	<p>Provide effort estimates for product features.</p> <p>Identify what features they can create in a given time frame — the sprint.</p> <p>Often provide time estimates for tasks in</p>	<p>Facilitate estimation poker games.</p> <p>Help development teams increase velocity, which affects time.</p> <p>Shield the team from</p>

	<p>Use velocity — development speed — to forecast long-term timelines.</p>	<p>Estimate effort for each sprint.</p> <p>Choose their own daily schedules and manage their own time.</p>	<p>Shield the team from organizational time-wasters and distractions.</p>
Cost	<p>Ultimately responsible for the budget and return on investment on an agile project.</p> <p>Use velocity to forecast long-term costs, based on timelines.</p>	<p>Provide effort estimates for product features.</p>	<p>Facilitate estimation poker games.</p> <p>Help development teams increase velocity, which affects cost.</p>
Team dynamics	<p>Commit to their projects as an integrated peer member of the scrum team.</p>	<p>Prevent bottlenecks by working cross-functionally, and are willing to take on different types of tasks.</p> <p>Continuously learn and teach one another.</p> <p>Commit, both individually and as part of the scrum team, to their projects and to one another.</p> <p>Strive to build consensus when making important decisions.</p>	<p>Facilitate scrum team collocation.</p> <p>Help remove impediments to scrum team self-management.</p> <p>Commit to their projects and are integrated members of the scrum team.</p> <p>Strive to build consensus within the scrum team when making important decisions.</p> <p>Facilitate relationships between the scrum team and stakeholders.</p>
Communication	<p>Communicate information about the product and the business needs to development teams on an ongoing basis.</p> <p>Communicate information about the project progress to stakeholders.</p> <p>Help present working functionality to stakeholders at the sprint review meetings at the end of each sprint.</p>	<p>Report on progress, upcoming tasks, and identify roadblocks in their daily scrum meetings.</p> <p>Keep the sprint backlog up-to-date daily, providing accurate, immediate information about a project's status.</p> <p>Present working functionality to project stakeholders at the sprint review meetings at the end of each sprint.</p>	<p>Encourage face-to-face communication between all scrum team members.</p> <p>Foster close cooperation between the scrum team and other departments within the company or organization.</p>
Stakeholders	<p>Set vision, release, and sprint goal expectations.</p> <p>Shield development team from business noise.</p> <p>Collect feedback during sprint reviews.</p> <p>Gather requirements throughout project.</p> <p>Communicate release dates and how new feature requests affect release dates.</p>	<p>Demonstrate working functionality at sprint reviews.</p> <p>Work through product owner to decompose requirements.</p> <p>Report on project progress through release and sprint burndown charts.</p> <p>Update task status no less than at the end of each day.</p>	<p>Coach on scrum and agile principles as they relate to their interaction with the scrum team.</p> <p>Shield developers from non-business distractions.</p> <p>Facilitate sprint reviews for gathering feedback.</p> <p>Facilitate interactions outside sprint reviews.</p>

Quality	<ul style="list-style-type: none"> <li>Add acceptance criteria to requirements.</li> <li>Ensure that the development team correctly understands and interprets requirements.</li> <li>Provide development teams with feedback about the product from the organization and from the marketplace.</li> <li>Accept functionality as done during each sprint.</li> </ul>	<ul style="list-style-type: none"> <li>Commit to providing technical excellence and good design.</li> <li>Test their work throughout the day and comprehensively test all development each day.</li> <li>Inspect their work and adapt for improvements at sprint retrospective meetings at the end of each sprint.</li> </ul>	<ul style="list-style-type: none"> <li>Help facilitate the sprint retrospective.</li> <li>Help ensure face-to-face communication between scrum team members, which in turn helps ensure quality work.</li> <li>Help create a sustainable development environment so that the development team can perform at its best.</li> </ul>
	<ul style="list-style-type: none"> <li>Look at overall project risks as well as risks to their ROI commitment.</li> <li>Prioritize high-risk items on the product backlog near the top to address them sooner rather than later.</li> </ul>	<ul style="list-style-type: none"> <li>Identify and develop the risk mitigation approach for each sprint.</li> <li>Alert the scrum master to roadblocks and distractions.</li> <li>Use information from each sprint retrospective to reduce risk in future sprints.</li> <li>Embrace cross-functionality to reduce risk if one member unexpectedly leaves the team.</li> <li>Commit to delivering shippable functionality at the end of each sprint, reducing risk in the overall project.</li> </ul>	<ul style="list-style-type: none"> <li>Help prevent roadblocks and distractions.</li> <li>Help remove roadblocks and identified risks.</li> <li>Facilitate development team conversations about possible risks.</li> </ul>



**REMEMBER** All in all, people on agile projects tend to find a great deal of job satisfaction. Self-management speaks to a deeply rooted human desire for autonomy — to control our own destiny — and allows people this control on a daily basis.

The next section discusses another reason that people on agile projects are happy: the servant-leader.

## ***Supporting the team: The servant-leader***

The scrum master serves as a servant-leader, someone who leads by removing obstacles, preventing distractions, and helping the rest of the scrum team do its job to the best of its ability. Leaders on agile projects help find

solutions rather than assign tasks. Scrum masters coach, trust, and challenge the scrum team to manage itself.

Other members of the scrum team can also take on servant leadership roles. While the scrum master helps get rid of distractions and roadblocks, the product owner and members of the development team can also help where needed. The product owner can lead by proactively providing important details about the product needs and quickly providing answers to questions from the development team. Development team members can teach and mentor one another as they become more cross-functional. Each person on a scrum team may act as a servant-leader at some point in the project.

Larry Spears identified ten characteristics of a servant-leader in his paper, “The Understanding and Practice of Servant-Leadership” (Servant Leadership Roundtable, School of Leadership Studies, Regent University, August 2005). Here are those characteristics, along with our additions for how each characteristic can benefit the team dynamics on an agile project.

- » **Listening:** Listening closely to other members of the scrum team will help the people on the scrum team identify areas to help one another. A servant-leader may need to listen to what people are saying, as well as what people are *not* saying, in order to remove obstacles.
- » **Empathy:** A servant-leader tries to understand and empathize with people on the scrum team, and to help them understand one another.
- » **Healing:** On an agile project, healing can mean undoing the damage of non-people-centric processes. These are processes that treat people like equipment and other replaceable parts. Many traditional project management approaches can be described as being non-people-centric.
- » **Awareness:** On an agile project, the people on the scrum team may need to be aware of activities on many levels to best serve the scrum team.
- » **Persuasion:** Servant-leaders rely on an ability to convince, rather than on top-down authority. Strong persuasion skills, along with organizational clout or influence, will help a scrum master advocate for the scrum team to the company or organization. A servant-leader can also pass along persuasion skills to the rest of the scrum team, helping maintain harmony and build consensus.
- » **Conceptualization:** Each member of a scrum team can use

conceptualization skills on an agile project. The changing nature of agile projects encourages the scrum team to envision ideas beyond those at hand. A servant-leader will help nurture the scrum team's creativity, both for the development of the product and for team dynamics.

- » **Foresight:** Scrum teams gain foresight with each sprint retrospective. By inspecting its work, processes, and team dynamics on a regular basis, the scrum team can continuously adapt and understand how to make better decisions for future sprints.
- » **Stewardship:** A servant-leader is the steward of the scrum team's needs. Stewardship is about trust. Members of the scrum team trust one another to look out for the needs of the team and the project as a whole.
- » **Commitment to the growth of people:** Growth is essential to a scrum team's ability to be cross-functional. A servant-leader will encourage and enable a scrum team to learn and grow.
- » **Building community:** A scrum team is its own community. A servant-leader will help build and maintain positive team dynamics within that community.

Servant leadership works because it positively focuses on individuals and interactions, a key tenet of agile project management. Much like self-management, servant leadership requires trust and respect.



TECHNICAL STUFF The concept of servant leadership is not specific to agile projects. If you have studied management techniques, you may recognize the works of Robert K. Greenleaf, who started the modern movement for servant-leadership — and coined the term *servant-leader* — in an essay in 1970. Greenleaf founded the Center for Applied Ethics, now known as the Greenleaf Center for Servant Leadership, which promotes the concept of servant leadership worldwide.

Another servant-leader expert, Kenneth Blanchard, co-wrote with Spencer Johnson the *One Minute Manager* (published by William Morrow), wherein he describes characteristics that make great managers of high-functioning people and teams. (The book has since been updated as *The New One-Minute*

*Manager*, published by Harper Collins India.) The reason the managers Blanchard studied were so effective is because they focused on ensuring that the people doing the work had direction, resources, and protection from noise to do their job as quickly as possible.

The next two sections largely relate to team factors for agile project success: the dedicated team and the cross-functional team.

## ***Working with a dedicated team***

Having a dedicated scrum team provides the following important benefits to projects:

- » **Keeping people focused on one project at a time helps prevent distractions.** Dedication to one project increases productivity by reducing *task-switching* — moving back and forth between different tasks without really completing any of them.
- » **Dedicated scrum teams have fewer distractions — and fewer distractions mean fewer mistakes.** When a person doesn't have to meet the demands of more than one project, that person has the time and clarity to ensure his or her work is the best it can be. [Chapter 15](#) discusses ways to increase product quality in detail.
- » **When people work on dedicated scrum teams, they know what they will be working on every day.** An interesting reality of behavioral science is that when people know what they will be working on in the immediate future, their minds engage those issues consciously at work and unconsciously outside the work environment. Stability of tasks engages your mind for much longer each day, enabling better solutions and higher quality products.
- » **Dedicated scrum team members are able to innovate more on projects.** When people immerse themselves in a product without distractions, they can come up with creative solutions for product functionality.
- » **People on dedicated scrum teams are more likely to be happy in their jobs.** By being able to concentrate on one project, a scrum team member's job is easier. Many, if not most, people enjoy producing quality work, being productive, and being creative. Dedicated scrum teams lead to higher satisfaction.

- » **When you have a dedicated scrum team working the same amount of time each week, you can accurately calculate *velocity* — the team's development speed.** In [Chapter 13](#), we talk about determining a scrum team's velocity at the end of each sprint and using velocity to determine long-term timelines and costs. Because velocity relies on comparing output from one sprint to the next, using velocity to forecast time and cost works best if the scrum team's work hours are constant. If you are unable to have a dedicated scrum team, at least try to have team members allocated to your project for the same amount of time each week.



TECHNICAL STUFF

The idea of the productive multitasker is a myth. In the past 25 years, and especially in the last decade, a number of studies have concluded that task-switching reduces productivity, impairs decision-making skills, and results in more errors.

To have a dedicated scrum team, you need strong commitment from your organization. Many companies ask employees to work on multiple projects at one time, under the mistaken assumption that the company will save money by hiring fewer people. When companies start to embrace a more agile mindset, they learn that the least expensive approach is to reduce defects and raise development productivity through focus.

Each member of the scrum team can help ensure dedication:

- » If you're a product owner, make sure that the company knows that a dedicated scrum team is a good fiscal decision. You are responsible for project return on investment, so be willing to fight for your project's success.
- » If you're a member of the development team and anyone requests that you do work outside the project, you can push back and involve the product owner or scrum master, if necessary. A request for outside work, regardless of how benign, is a potential roadblock.
- » If you're a scrum master, as the expert on agile approaches, you can educate the company on why a dedicated scrum team means increased productivity, quality, and innovation. A good scrum master should also

have the organizational clout to keep the company from poaching people from the scrum team for other projects.

Another characteristic of scrum teams is that they are cross-functional.

## ***Working with a cross-functional team***

Cross-functional development teams are also important on agile projects. The development team on an agile software project doesn't just include programmers; it could include all the people who will have a job on the project. For example, a development team on a software project might include programmers, database experts, quality assurance people, usability experts, and graphic designers. While each person has specialties, being cross-functional means that everyone on the team is willing to pitch in on different parts of the project, as much as possible.

On an agile development team, you continuously ask yourself two questions: "What can I contribute today?" and "How can I expand my contribution in the future?" Everyone on the development team will use his or her current skills and specialties in each sprint. Cross-functionality gives development team members the opportunity to learn new skills by working on areas outside of their expertise. Cross-functionality also allows people to share their knowledge with their fellow development team members. You don't need to be a jack-of-all-trades to work on an agile development team, but you should be willing to learn new skills and help with all kinds of tasks.



TECHNICAL STUFF Although task-switching decreases productivity, cross-functionality works because you're not changing the context of what you are working on; you're looking at the same problem from a different perspective. Working on different aspects of the same problem increases knowledge depth and your ability to do a better job.

The biggest benefit of a cross-functional development team is the elimination of single points of failure. If you have worked on a project before, how many times have you experienced delays because a critical member of the team is on vacation, out sick, or, worse, has left the company? Vacations, illness, and turnover are facts of life, but with a cross-functional development team, other

team members can jump in and continue work with minimal disruption. Even if an expert leaves the project team unexpectedly and abruptly, other development team members will know enough about the work to keep it progressing.



**WARNING** Development team members go on vacation or catch the flu. Don't sabotage your project by having only one person know a skill or functional area.

Cross-functionality takes strong commitment from the development team, both as individual members and as a group. The old phrase, “There is no *i* in *team*” is especially true on agile projects. Working on an agile development team is about skills, rather than titles.



**TIP** Development teams without titles are more merit-based because team seniority and status is based on current knowledge, skills, and contribution.

Letting go of the idea that you're a “senior quality assurance tester” or a “junior developer” can require a new way of thinking about yourself. Embracing the concept of being part of a cross-functional development team may take some work, but it can be rewarding as you learn new skills and develop a rhythm of teamwork.



**TIP** When developers also test, they create code that is test-friendly.

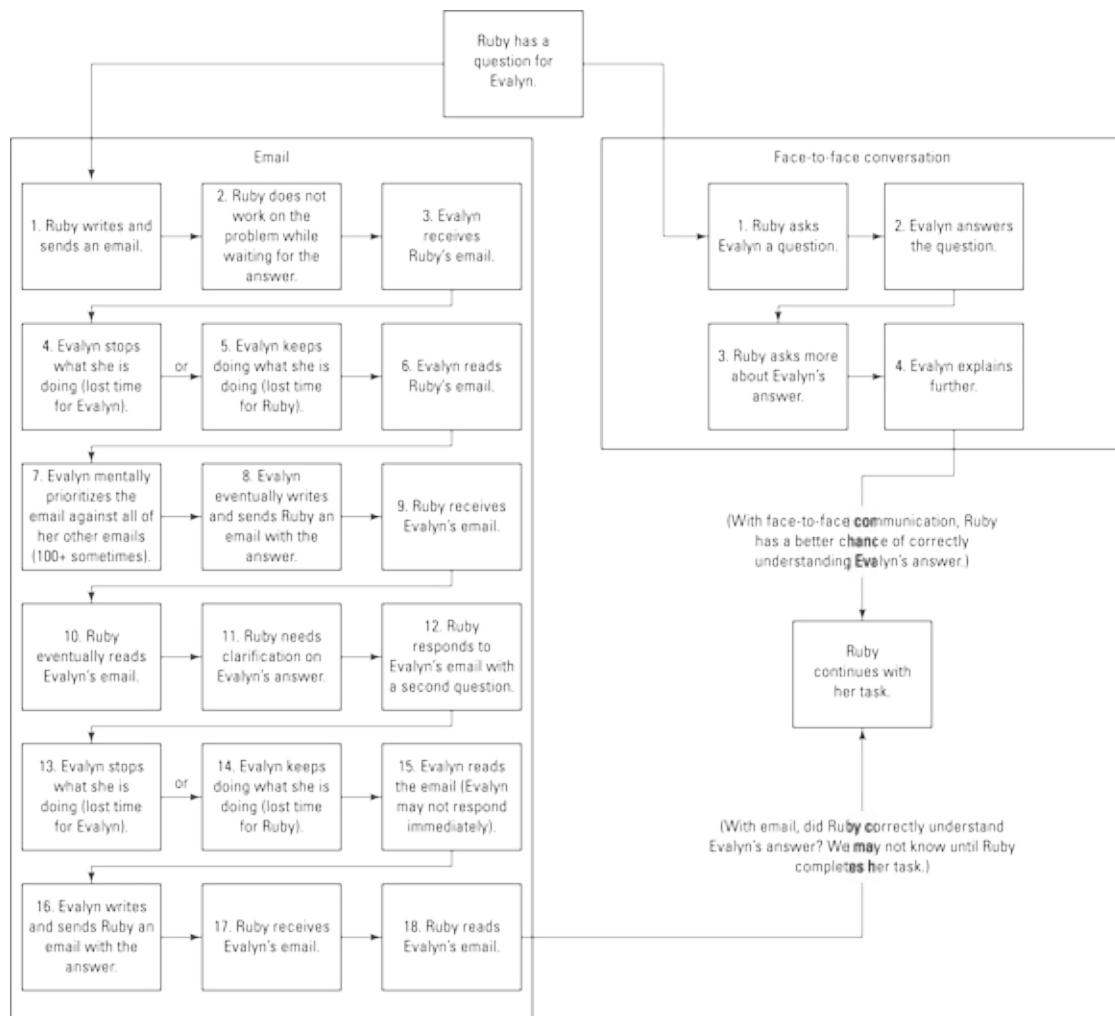
Having a cross-functional development team also requires commitment and support from your organization. Some companies eliminate titles or keep them intentionally vague (you might see something like “application development”) to encourage teamwork. Other techniques for creating a strong cross-functional development team from an organizational standpoint include offering training, recognizing scrum teams as a whole, and being willing to make changes if a particular person does not fit in with a team environment.

When hiring, your company can actively look for people who will work well in a highly collaborative environment, who want to learn new tasks, and who are willing to work on all areas of a project.

Both the physical environment and the cultural environment of an organization are important keys to success with agile projects. The next section shows you how.

## ***Reinforcing openness***

As we explain in other chapters, a collocated scrum team is ideal. The Internet has brought people together globally, but nothing — not the best combination of emails, instant messages, videoconferencing, phone calls, and online collaboration tools — can replace the simplicity and effectiveness of a face-to-face conversation. [Figure 14-1](#) illustrates the difference between an email exchange and a conversation in person.



#### **FIGURE 14-1:** Email versus face-to-face conversation.

The idea of scrum team members working in the same physical location and being able to talk in person, instantly, is important to team dynamics. You find more details on communication later in this chapter. Also, [Chapter 5](#) provides details on how to set up the physical environment for a scrum team.

Having a cultural environment of openness, which is conducive to scrum team growth, is another success factor for agile projects. Everyone on a scrum team should be able to

- » Feel safe.
- » Speak his or her mind in a positive way.
- » Challenge the status quo.
- » Be open about challenges without being penalized.
- » Request resources that will make a difference to the project.
- » Make mistakes and learn from them.
- » Suggest change and have other scrum team members seriously consider those changes.
- » Respect fellow scrum team members.
- » Be respected by other members of the scrum team.

Trust, openness, and respect are fundamental to team dynamics on an agile project.



**TIP** Some of the best product and process improvements come from novices asking “silly” questions.

Another facet of agile team dynamics is the concept of the size-limited team.

### ***Limiting development team size***

An interesting psychological aspect of team dynamics on an agile project is the number of people on a development team. Development teams usually have between three and nine people. An ideal size is somewhere in the

middle.

Limiting development team size to this range provides a team with enough diverse skills to take a requirement from paper to production while keeping communication and collaboration simple. Development team members can easily interact with one another and make decisions by consensus.

When you have development teams with more than nine people, the people on those teams tend to break into subgroups and build silos. This is normal social human behavior, but subgroups can be disruptive to a development team striving to be self-managing. It is also more difficult to communicate with larger development teams; there are more communication channels and opportunities to lose or misconstrue a message. With more than nine people on a development team, you often need an extra person just to help manage communication.

Development teams with fewer than nine people, on the other hand, tend to naturally gravitate to an agile approach. However, development teams that are too small may find working cross-functionally difficult because there may not be enough people with varying skills on the project.



**TIP** If your product development requires more than nine development team members, consider breaking up the work between multiple scrum teams. Creating teams of people with similar personalities, skills, and work styles can improve productivity. Find details on how to work with multiple scrum teams in [Chapters 13](#) and [17](#).

## ***Managing projects with dislocated teams***

As we say throughout the book, a collocated scrum team is ideal for agile projects. However, sometimes it isn't possible for a scrum team to work together in one place. *Dislocated teams*, teams with people who work in different locations, exist for many reasons and in different forms.

In some companies, the people with the right skills for a project may work in different offices, and the company may not want the cost of bringing those people together for the project's duration. Some organizations work jointly with other organizations on projects, but may not want or be able to share office space. Some people may telecommute, especially contractors, live long

distances from the company they work with, and never visit that company's office. Some companies work with offshore groups and create projects with people from other countries.

The good news is that you can still have an agile project with a dislocated scrum team or teams. If you have to work with a dislocated team, we've found that an agile approach allows you to see working functionality much sooner and limits the risk of inevitable misunderstandings that a dislocated team will experience.

In *A Scrum Handbook* (Scrum Institute Training Press), Jeff Sutherland describes three models of distributed scrum teams:

- » **Isolated scrums:** With isolated scrums, individual scrum teams have collocated scrum team members, but each scrum team is in a separate geographic location and works separately. Product development with isolated scrums has only code-level integration; that is, the different teams don't communicate or work together but expect the code to work when it is time to integrate each module due to organizational coding standards. Isolated scrums tend to struggle because different people interpret coding standards differently.
- » **Distributed scrum of scrums:** With a distributed scrum of scrums model, scrum teams are in different locations, like in isolated scrums. To coordinate work, scrum teams hold a *scrum of scrums* — a meeting of multiple scrum masters — to integrate on a daily basis.
- » **Integrated scrums:** Integrated scrum teams are cross-functional, with scrum team members in different locations. A scrum of scrums still occurs but face-to-face communication is lost.

[Table 14-3](#), from Ambyssoft's "Agile Adoption Rate Survey Results" in 2008, shows a comparison of success rates for projects with collocated scrum teams against those with geographically dispersed scrum teams.

**TABLE 14-3 Success of Collocated and Dislocated Scrum Teams**

<i>Team Location</i>	<i>Success Percentage</i>
Collocated scrum team	83%

Source: Ambyssoft, Agile Adoption Rate Survey Results, 2008

Dislocated but physically reachable 72%

Distributed across geographies 60%

*"Agile Adoption Rate Survey Results" (Scott W. Ambler, Amblysoft, Copyright © 2008)*

How do you have a successful agile project with a dislocated scrum team? We have three words: communicate, communicate, and communicate. Because daily in-person conversations are not possible, agile projects with dislocated scrum teams require unique efforts by everyone working on the project. Here are some tips for successful communication among non-collocated scrum team members:

- » **Use videoconferencing technology to simulate face-to-face conversations.** The majority of interpersonal communication is visual, involving facial cues, hand gestures, and even shoulder shrugs. Videoconferencing enables people to see one another and benefit from nonverbal communication as well as a discussion. Use videoconferencing, or even telepresence robots, liberally throughout the day, not just for sprint meetings. Make sure team members are ready for impromptu video chats, and that the technology makes it easy to initiate them.
- » **If possible, arrange for the scrum team members to meet in-person in a central location at least once at the beginning of the project, and preferably multiple times throughout the project.** The shared experience of meeting in-person, even once or twice, can help build teamwork among dislocated team members. Working relationships built through face-to-face visits are stronger and carry on after the visit ends.
- » **Use an online collaboration tool.** Some tools simulate whiteboards and user story cards, track conversations, and enable multiple people to update artifacts at the same time.
- » **Include scrum team members' pictures on online collaboration tools, or even in email address signature lines.** Humans respond to faces more than written words alone. A simple picture can help humanize instant messages and emails.
- » **Be cognizant of time zone differences.** Put multiple clocks showing different time zones on the wall so you don't accidentally call someone's cellphone at 3 a.m. and wake up that person — or wonder why he or she

isn't answering.

- » **Be flexible because of time zone differences as well.** You may need to take video calls or phone calls at odd hours from time to time to help keep project work moving. For drastic time zone differences, consider trading off on times you are available. One week, Team A can be available in the early morning. The next, Team B can be available later in the evening. That way, no one always has an inconvenience.
- » **If you have any doubt about a conversation or a written message, ask for clarification by phone or video.** It always helps to double-check when you're unsure of what someone meant. Follow up with a call to avoid mistakes from miscommunication.
- » **Be aware of language and cultural differences between scrum team members, especially when working with groups in multiple countries.** Understanding colloquialisms and pronunciation differences can increase the quality of your communication across borders. It helps to know about local holidays, too. We've been blindsided more than once by closed offices outside our region.
- » **Make an extra attempt to discuss non-work topics sometimes.** Discussing non-work topics helps you grow closer to scrum team members, regardless of location.

With dedication, awareness, and strong communication, distributed agile projects can succeed.

The unique approaches to team dynamics on agile projects are part of what make agile projects successful. Communication is closely related to team dynamics, and the communication methods on agile projects also have big differences from traditional projects, as you see in the following section.

## ***What's Different about Agile Communication?***

Communication, in project management terms, is the formal and informal ways the people on the project team convey information to each other. As with traditional projects, good communication is a necessity for agile

projects.

However, the agile principles set a different tone for agile projects, emphasizing simplicity, directness, and face-to-face conversations. The following agile principles relate to communication:

4. Business people and developers must work together daily throughout the project.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
10. Simplicity — the art of maximizing the amount of work not done — is essential.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The Agile Manifesto also addresses communication, valuing working software over comprehensive documentation. Although documentation has value, working functionality has more importance on an agile project.

[Table 14-4](#) shows some differences between communication on traditional projects and on agile projects.

## **TABLE 14-4 Traditional versus Agile Communication**

<i>Communication Management with Traditional Approaches</i>	<i>Communication Management with Agile Approaches</i>
Team members might make no special effort for in-person conversations.	Agile project management approaches value face-to-face communication as the best way to convey information.
Traditional approaches place high value on documentation. Teams may create a large number of complex documents and status reports based on process, rather than considering actual need.	Agile documents, or <i>artifacts</i> , are intentionally simple and provide information that is barely sufficient. Agile artifacts only contain essential information and can often convey project status at a glance. Project teams use the <i>show, don't tell</i> concept, showing working software to communicate progress on a regular basis in the sprint review.
Team members may be required to attend a large number of meetings, whether or not those meetings are useful or necessary.	Meetings on agile projects are, by design, as quick as possible and include only people who will add to the meeting and benefit from the meeting. Agile meetings provide all the benefits of face-to-face communication without wasting time. The structure of agile meetings is to enhance, not reduce, productivity.



**TIP** The question of how much documentation is required is not a volume question but an appropriateness question. Why do you need a specific document? How can you create it in the simplest way possible? You can use poster-sized sticky sheets to put on the wall and make information digestible. This can also work best for visually conveying artifacts such as the vision statement, the definition of done, the impediments log, and important architectural decisions. Pictures truly are worth a thousand words.

The following sections show how to take advantage of the agile framework's emphasis on in-person communication, focus on simplicity, and value of working functionality as a communication medium.

## ***Managing Agile Communication***

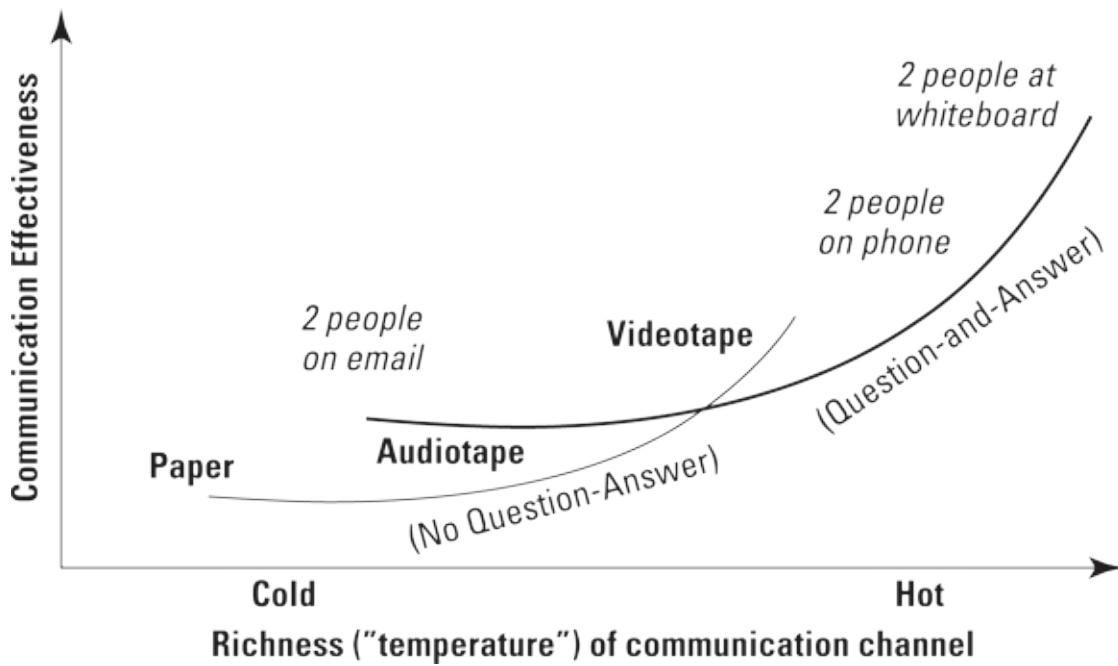
To manage communication on agile projects, you need to understand how different agile communication methods work and how to use them together. You also need to know why status on an agile project is different and how to report project progress to stakeholders. The following sections show you how.

### ***Understanding agile communication methods***

You can communicate on an agile project through artifacts, meetings, and informally.

Face-to-face conversations are the heart and soul of agile projects. When scrum team members talk with one another about the project throughout every day, communication is easy. Over time, scrum team members understand each other's personality, communication style, and thought processes, and will be able to communicate quickly and effectively.

[Figure 14-2](#), from Alistair Cockburn's presentation *Software Development as a Cooperative Game*, shows the effectiveness of face-to-face communication versus other types of communication.



Copyright © Humans and Technology, Inc.

**FIGURE 14-2:** Comparison of communication types.

In previous chapters, we describe a number of artifacts and meetings that fit with agile projects. All the agile artifacts and meetings play a role in communication. Agile meetings provide a format for communicating in a face-to-face environment. Meetings on agile projects have a specific purpose and a specific amount of time so that the development team can work, rather than sit in meetings. Agile artifacts provide a format for written communication that is structured but not cumbersome or unnecessary.

[Table 14-5](#) provides a view of the different communication channels on an agile project.

## TABLE 14-5 Agile Project Communication Channels

<i>Channel</i>	<i>Type</i>	<i>Role in Communication</i>
Project planning, release planning, and sprint planning	Meetings	Planning meetings have specific desired outcomes and concisely communicate the purpose and details of the project, the release, and the sprint to the scrum team. Learn more about planning meetings in <a href="#">Chapters 7 and 8</a> .
Product vision statement	Artifact	The product vision statement communicates the end goal of the project to the project team and the organization. Find out more about the product vision in <a href="#">Chapter 7</a> .
Product roadmap	Artifact	The product roadmap communicates a long-term view of the features that

roadmap	Artifact	support the product vision and are likely to be part of the project. Find out more about the product roadmap in <a href="#">Chapter 7</a> .
Product backlog	Artifact	The product backlog communicates the scope of the project as a whole to the project team. Find out more about the product backlog in <a href="#">Chapters 7 and 8</a> .
Release plan	Artifact	The release plan communicates the goals and timing for a specific release. Find out more about the release plan in <a href="#">Chapter 8</a> .
Sprint backlog	Artifact	When updated daily, the sprint backlog provides immediate sprint and project status to anyone who needs that information. The burndown chart on the sprint backlog provides a quick visual of the sprint progress. Find out more about the sprint backlog in <a href="#">Chapters 8 and 9</a> .
Task board	Artifact	Using a task board visually radiates the status of the current sprint or release to anyone who walks by the scrum team's work area. Find out more about the task board in <a href="#">Chapter 9</a> .
Daily scrum	Meeting	The daily scrum provides the scrum team with a verbal, face-to-face opportunity to coordinate the priorities of the day and identify any challenges. Find out more about daily scrum meetings in <a href="#">Chapter 9</a> .
Face-to-face conversations	Informal	Face-to-face conversations are the most important mode of communication on an agile project.
Sprint review	Meeting	The sprint review is the embodiment of show, don't tell, philosophy. Displaying working functionality to the entire project team conveys project progress in a more meaningful way than a written report or a conceptual presentation ever could. Find out more about sprint reviews in <a href="#">Chapter 10</a> .
Sprint retrospective	Meeting	The sprint retrospective allows the scrum team to communicate with one another specifically for improvement. Find out more about sprint retrospectives in <a href="#">Chapter 10</a> .
Meeting notes	Informal	Meeting notes are an optional, informal communication method on an agile project. Meeting notes can capture action items from a meeting to ensure that people on the scrum team remember them for later. Notes from a sprint review include new features for the product backlog. Notes from a sprint retrospective can remind the scrum team of plans for improvement.
Collaborative solutions	Informal	Whiteboards, sticky notes, and electronic collaboration tools all help the scrum team communicate. Ensure that these tools augment, rather than replace, face-to-face conversations. Capturing and saving collaboration results is a low-fidelity way to remind the team of decisions made for immediate and future consideration.



**REMEMBER** Artifacts, meetings, and more informal communication channels are all tools. Keep in mind that even the best tools need people to use those tools correctly to be effective. Agile projects are about people and interactions; tools are secondary to success.

The next section addresses a specific area of agile project communication: status reporting.

## **Status and progress reporting**

All projects have stakeholders, people outside the immediate scrum team who have a vested interest in the project. At least one of the stakeholders is the person responsible for paying for your project (the project sponsor). It is important for stakeholders, especially those responsible for budgets, to know how the project is progressing. This section shows how to communicate your project's status.

*Status* on an agile project is a measure of the features that the scrum team has completed. Using the definition of done from [Chapters 2, 8, 10](#), and [15](#), a feature is complete if the scrum team has developed, tested, integrated, and documented that feature, per the agreement between the product owner and the development team.

If you've worked on a traditional software project, how many times have you been in a status meeting and reported that the project was, say, 64 percent complete? If your stakeholders had replied, "Great! We would like that 64 percent now; we ran out of funds," you and the stakeholders alike would be at a loss, because you didn't mean that 64 percent of your features were ready to use. You meant that each one of the product features was only 64 percent in progress, you had no working functionality, and you still had a lot of work to do before anyone could use the product.

On an agile project, working functionality that meets the definition of done is the primary measure of progress. You can confidently say that project features are complete. Because scope changes constantly on agile projects, you would not express status as a percentage. Instead, a list of potentially shippable features would be more interesting for stakeholders to see as it grows.

Track the progress of your sprint and the project daily. Your primary tools for communicating status and progress are the task board, sprint backlog, product backlog, release and sprint burndown charts, and the sprint review.



TIP The sprint review is where you demonstrate working software to your project stakeholders. Resist creating slides or handouts; the key to the sprint review is showing your stakeholders progress as a demonstration,

rather than only telling them what you completed. Show, don't tell.

Strongly encourage anyone who may have an interest in your project to come to your sprint reviews. When people see the working functionality in action, especially on a regular basis, they get a much better sense of the work you've completed.



**WARNING** Companies and organizations that are starting out using agile techniques may expect to see traditional status reports, in addition to agile artifacts. These organizations may also want members of the scrum team to attend regular status meetings, outside of the daily scrums and other agile meetings. This is called *double work agile*, because you are doing twice as much work as necessary. Double work agile is one of the top pitfalls for agile projects. Scrum teams will burn out quickly if they try to meet the demands of two drastically different project approaches. You can avoid double work agile by educating your company about why agile artifacts and events are a better replacement for old documents and meetings. Insist on experimenting with agile artifacts and events to conduct a successful agile project.

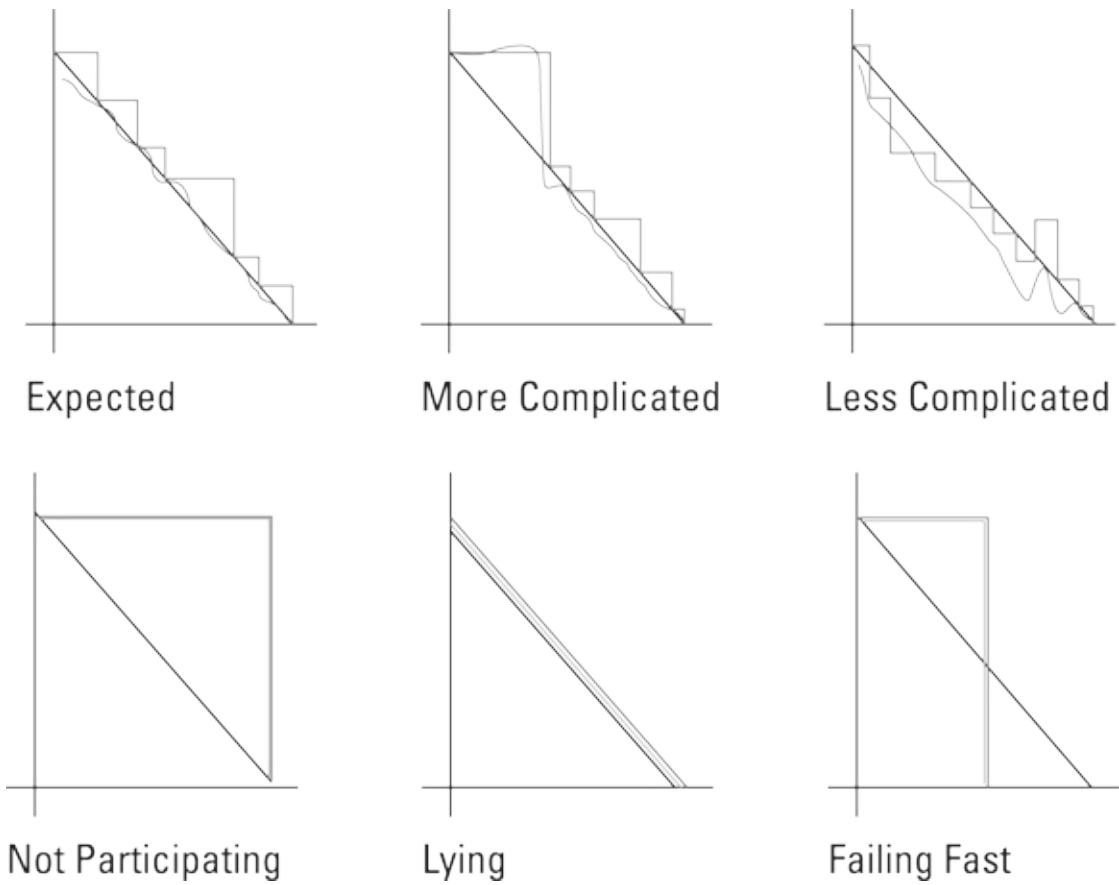
The sprint backlog is a report of the daily status of your current sprint. The sprint backlog contains the sprint's user stories and their related tasks and estimates. The sprint backlog also often has a burndown chart that visually shows the status of the work the development team has completed and the remaining work to complete the requirements in the sprint. The development team is in charge of updating the sprint backlog at least once a day by updating the number of hours of work remaining for each task.



**WARNING** If you're a project manager now, or if you study project management in the future, you may come across the concept of *earned value management* (EVM), as a way of measuring project progress and performance. Some agile practitioners try to use an agile-like version of EVM, but we avoid EVM for agile projects. EVM assumes that your project has a fixed scope, which is antithetical to an agile approach.

Instead of trying to change agile approaches to fit into old models, use the tools here — they work.

The burndown chart quickly shows, rather than tells, status. When you look at a sprint burndown chart, you can instantly see whether the sprint is going well or might be in trouble. In [Chapter 9](#), we show you an image of sample burndown charts for different sprint scenarios; here it is again in [Figure 14-3](#).



[FIGURE 14-3:](#) Profiles of burndown charts.

If you update your sprint backlog every day, you'll always have an up-to-date status for your project stakeholders. You can also show them the product backlog so that they know which features the scrum team has completed to date, which features will be part of future sprints, and the priority of the features.



**REMEMBER** The product backlog will change as you add and reprioritize features.

Make sure that people who review the product backlog, especially for status purposes, understand this concept.



TIP A task board is a great way to quickly show your project team the status of a sprint, release, or even of the entire project. Task boards have sticky notes with user story titles in at least four columns: To Do, In Progress, Accept, and Done. If you display your task board in the scrum team's work area, anyone who walks by can see a high-level status of which product features are done and which features are in progress. The scrum team always knows where the project stands, because the scrum team sees the task board every day.

Always strive for simple, low-fidelity information radiators to communicate status and progress. The more you can make information accessible and on-demand, the less time you and your stakeholders will spend preparing and wondering about status.

# Chapter 15

## Managing Quality and Risk

---

### IN THIS CHAPTER

- » Learning how agile project management quality approaches reduce risk
- » Discovering ways to ensure quality development
- » Taking advantage of automated testing for better productivity
- » Understanding how agile project approaches reduce risk

Quality and risk are closely related parts of project management. In this chapter, you find out how to deliver quality products using agile project management methods. You understand how to take advantage of agile approaches to manage risk on your projects. You see how quality has historically affected project risk, and how quality management on agile projects fundamentally reduces project risk.

### ***What's Different about Agile Quality?***

*Quality* refers to whether a product works, and whether it fulfills the project stakeholders' needs. Quality is an inherent part of agile project management. All 12 agile principles that we list in [Chapter 2](#) promote quality either directly or indirectly. Those principles follow:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the

project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity — the art of maximizing the amount of work not done — is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

These principles emphasize creating an environment where agile teams are able to produce valuable, working functionality. Agile approaches encourage quality both in the sense of products working correctly and meeting the needs of project stakeholders.

[Table 15-1](#) shows some differences between quality management on traditional projects and on agile projects.

## **TABLE 15-1 Traditional versus Agile Quality**

<i>Quality Management with Traditional Approaches</i>	<i>Quality Dynamics with Agile Approaches</i>
Testing is the last phase of a project before product deployment. Some features are tested months after they were created.	Testing is a daily part of each sprint and is included in each requirement's definition of done. You use automated testing, allowing quick and robust testing every day.
Quality is often a reactive practice, with the focus mostly on product testing and issue resolution.	You address quality both reactively, through testing, and proactively, encouraging practices to set the stage for quality work. Examples of proactive quality approaches include face-to-face communication, pair programming, and established coding standards.

---

Problems are riskier when found at the end of a project. Sunk costs are high by the time teams reach testing.

You can create and test riskier features in early sprints, when sunk costs are still low.

---

Problems or defects, sometimes called *bugs* in software development, are hard to find at the end of a project, and fixes for problems at the end of a project are costly.

Problems are easy to find when you test a smaller amount of work. Fixes are easier when you fix something you just created, rather than something you created months earlier.

---

Sometimes, to meet a deadline or save money, teams cut the testing phase short.

Testing is assured on agile projects because it is part of every sprint.

---

At the start of this chapter, we state that quality and risk are closely related. The agile approaches in [Table 15-1](#) greatly reduce the risk and unnecessary cost that usually accompany quality management.

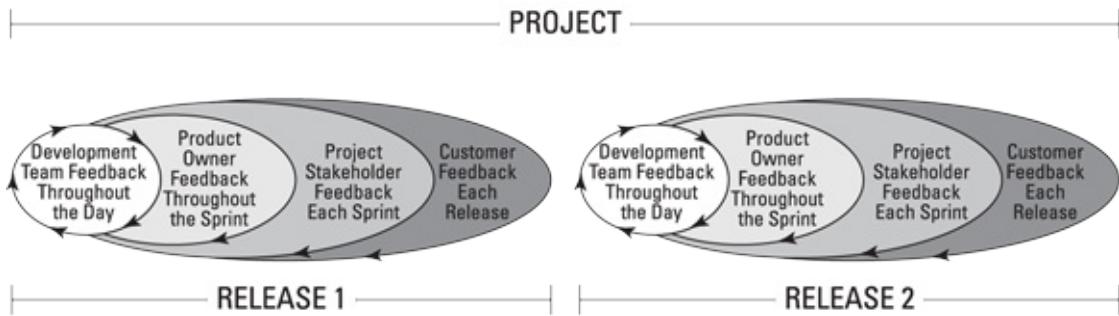


TECHNICAL  
STUFF

## BUGS. BUGS? BUGS!

Why do we call computer problems *bugs*? The first computers were large, glass-encased machines that took up entire rooms. In 1945, one of these behemoth computers, the Mark II Aiken Relay Calculator at Harvard University, had problems with one of its circuits. Engineers traced the issue to a moth — a literal bug — in the machine. After that, the team's running joke was that any issue with the computer had to be a bug. The term stuck, and people still use *bug* today to describe hardware problems, software problems, and sometimes even problems outside of the computer science realm. The engineers at Harvard even taped the moth to a logbook. That first bug is now on display at the Smithsonian National Museum of American History.

Another difference about quality on agile projects is the multiple quality feedback loops throughout a project. In [Figure 15-1](#), you see the different types of product feedback a scrum team receives in the course of a project. The development team can immediately incorporate this feedback into the product, increasing product quality on a regular basis.



**FIGURE 15-1:** Quality feedback in an agile project.



**REMEMBER** In [Chapter 14](#), we tell you that development teams on agile projects can include everyone who works on a product. Development teams on agile projects typically include people who are experts in creating and executing tests and ensuring quality. Development team members are cross-functional; that is, every team member may do different jobs at different times during the project. Cross-functionality extends to quality activities such as preventing issues, testing, and fixing bugs.

In the next section, you see how to use agile project management techniques to increase quality.

## Managing Agile Quality

Agile development teams have the primary responsibility for quality on agile projects. The responsibility for quality is an extension of the responsibilities and freedoms that come with self-management. When the development team is free to determine its development methods, the development team is also responsible for ensuring that those methods result in quality work.



**TECHNICAL STUFF** Organizations often refer to quality management as a whole as *quality assurance*, or *QA*. You may see QA departments, QA testers, QA managers, QA analysts, and all other flavors of QA-prefix titles to refer to people who are responsible for quality activities. QA is also sometimes used as shorthand for testing, as in “we performed QA on the

product” or “now we are in the QA phase.” Quality control (QC) is also a common way to refer to quality management.

The other members of the scrum team — the scrum master and the product owner — also play parts in quality management. Product owners provide clarification on requirements and also accept those requirements as being done throughout each sprint. Scrum masters help ensure development teams have a work environment where the people on development teams can work to the best of their abilities.

Luckily, agile project management approaches have several ways to help scrum teams create quality products. In this section, you see how testing in sprints increases the likelihood of finding defects and reduces the cost of fixing them. You gain an understanding of the many ways agile project management proactively encourages quality product development. You see how inspecting and adapting on a regular basis addresses quality. Finally, you find out how automated testing is essential to delivering valuable products continuously throughout an agile project.

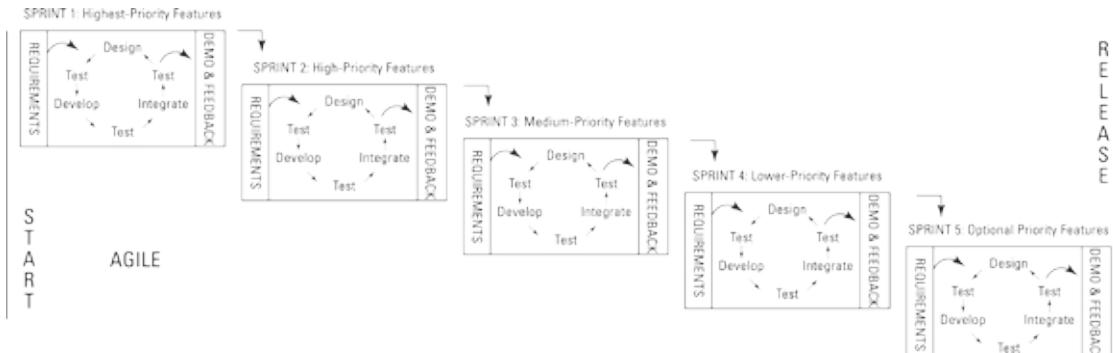
## ***Quality and the sprint***

Quality management is a daily part of agile projects. Scrum teams run agile projects in sprints, short development cycles that last one to four weeks. Each cycle includes activities from the different phases of a traditional project for each user story in the sprint: requirements, design, development, testing, and integration for deployment. Find out more about working in sprints in [Chapters 8, 9, and 10](#).



**TIP** Here’s a quick riddle: Is it easier to find a quarter on a table or in a stadium? Obviously, the answer is a table. Just as obvious is that it is easier to find a defect in 100 lines of software code than in 100,000 lines. Iterative development makes quality product development easier.

Scrum teams test throughout each sprint. [Figure 15-2](#) shows how testing fits into sprints on an agile project. Notice that testing begins in the first sprint, right after developers start creating the first requirement in the project.



**FIGURE 15-2:** Testing within sprints.

When development teams test throughout each sprint, they can find and fix defects very quickly. With agile project management, development teams create product requirements, immediately test those requirements, and fix any problems immediately before considering the work done. Instead of trying to remember how to fix something they created weeks or months ago, development teams are, at the most, fixing the requirement they worked on one or two days earlier.

Testing every day on an agile project is a great way to ensure product quality. Another way to ensure product quality is to create a better product from the start. The next section shows you different ways that agile project management helps you avoid errors and create an excellent product.

## ***Proactive quality***

An important and often-neglected aspect of quality is the idea of preventing problems. A number of agile approaches allow and encourage scrum teams to proactively create quality products. These practices include

- » An emphasis on technical excellence and good design
- » Incorporation of quality-specific development techniques into product creation
- » Daily communication between the development team and the product owner
- » Acceptance criteria built into user stories
- » Face-to-face communication and collocation
- » Sustainable development
- » Regular inspection and adaption of work and behavior

The following sections provide a detailed look at each of these proactive quality practices.



**REMEMBER** Quality means both that a product works correctly and that the product does what the project stakeholders need it to do.

### ***Continuous attention to technical excellence and good design***

Agile teams focus on technical excellence and good design because these traits lead to valuable products. How do development teams provide great technical solutions and designs?

One way that development teams provide technical excellence is through self-management, which provides them with the freedom to innovate technically. Traditional organizations may have mandatory technical standards that may or may not make sense for a given project. Self-organizing development teams have the freedom to decide whether a standard will provide value in creating a product, or if a different approach will work better. Innovation can lead to good design, technical excellence, and product quality.

Self-management also provides development teams with a sense of product ownership. When people on development teams feel a deep responsibility for the product they're creating, they often strive to find the best solutions and execute those solutions in the best way possible.



**TIP** Nothing is more sophisticated than a simple solution.

Organizational commitment also plays a role in technical excellence. Some companies and organizations, regardless of their project management approaches, have a commitment to excellence. Think about the products that you use every day and associate with quality; chances are those products come from companies that value good technical solutions. If you're working on an agile project for a company that believes in and rewards technical excellence, enacting this agile principle will be easy.

Other companies may undervalue technical excellence; agile project teams at these companies may struggle when trying to justify training or tools that will help create better products. Some companies do not make the connection between good technology, good products, and profitability. Scrum masters and product owners may need to educate their companies on why good technology and design are important and may need to lobby to get development teams what they need to create a great product.



**WARNING** Don't confuse technical excellence with using new technologies for the sake of using something new or trendy. Your technology solutions should efficiently support the product needs, not just add to a resume or a company skills profile.

By incorporating technical excellence and good design into your everyday work, you create a quality product that you are proud of.

### ***Quality development techniques***

During the past several decades of software development, the motivation to be more adaptive and agile has inspired a number of agile development techniques that focus on quality. This section provides a high-level view of a few extreme programming (XP) development approaches that help ensure quality proactively. For more information on XP practices, see [Chapter 4](#).



**TIP** Many agile quality management techniques were created with software development in mind. You can adapt some of these techniques when creating other types of products, such as hardware products or even building construction. If you're going to work on a non-software project, read about the development methods in this section with adaptability in mind:

- » **Test-driven development (TDD):** This development method begins with a developer creating a test for the requirement he or she wants to create. The developer then runs the test, which should fail at first because the functionality does not yet exist. The developer develops until the test

passes, and then refactors the code — takes out as much code as possible, while still having the test pass. With TDD, you know that the newly created functionality of a requirement works correctly because you test while you create the functionality and develop the functionality until the test passes.

- » **Pair programming:** With *pair programming*, developers work in groups of two. Both developers sit at the same computer and work as a team to create one product requirement. The developers take turns at the keyboard to collaborate. Usually, the one at the keyboard takes a direct tactical role, while the observing partner takes a more strategic or navigating role, looking ahead and providing in-the-moment feedback. Because the developers are literally looking over one another's shoulder, they can catch errors quickly. Pair programming increases quality by providing instant error checks and balances.
- » **Peer review:** Sometimes called *peer code review*, a *peer review* involves members of the development team reviewing one another's code. Like pair programming, peer reviews have a collaborative nature; when developers review each other's finished products, the developers work together to provide solutions for any issues they find. If development teams don't practice pair programming, they should at least practice peer reviews, which increase quality by allowing development experts to look for structural problems within product code.
- » **Collective code ownership:** In this approach, everyone on the development team can create, change, or fix any part of the code on the project. Collective code ownership can speed up development, encourage innovation, and with multiple pairs of eyes on the code, help development team members quickly find defects.
- » **Continuous integration:** This approach involves the creation of integrated code builds one or more times each day. Continuous integration allows members of the development team to check how the user story that the development team is creating works with the rest of the product. Continuous integration helps ensure quality by allowing the development team to check for conflicts regularly. Continuous integration is essential to automated testing on agile projects; you need to create a code build at the end of the day before running automated tests overnight. Find out more about automated testing later in this chapter.



**REMEMBER** On an agile project, the development team decides which tools and techniques will work best for the project, the product, and the individual development team.

Many agile software development techniques help ensure quality, and there is a lot of discussion and information about these techniques in the community of people who use agile project management approaches. We encourage you to learn more about these approaches if you’re going to work on an agile project, especially if you’re a developer. Entire books are dedicated to some of these techniques, such as test-driven development. The information we provide here is at the tip of the iceberg. See [Chapter 22](#) for more recommendations.

### ***The product owner and development team***

Another aspect of agile project management that encourages quality is the close relationship between the development team and the product owner. The product owner is the voice of business needs for the product. In this role, the product owner works with the development team every day to ensure that the functionality meets those business needs.

During planning stages, the product owner’s job is to help the development team understand each requirement correctly. During the sprint, the product owner answers questions that the development team has about requirements and is responsible for reviewing functionality and accepting them as done. When the product owner accepts requirements, he or she ensures that the development team correctly interpreted the business need for each requirement, and that the new functionality performs the task that it needs to perform.

In waterfall projects, feedback loops between developers and business owners are less frequent, so a development team’s work typically strays from the original product goals set in the product vision statement.

A product owner who reviews requirements daily catches misinterpretations early. The product owner can then set the development team back on the right path, avoiding a lot of wasted time and effort.



**REMEMBER** The product vision statement communicates how your product supports the company's or organization's strategies. The vision statement articulates the product's goals. [Chapter 7](#) explains how to create a product vision statement.

### **User stories and acceptance criteria**

Another proactive quality measure on agile projects is the acceptance criteria you build into each user story. In [Chapter 7](#), we explain that a user story is one format for describing product requirements. User stories increase quality by outlining the specific actions the user will take to correctly meet business needs. [Figure 15-3](#) shows a user story and its acceptance criteria.

<b>Title</b> Transfer money between accounts	<b>When I do this:</b> When I view my account balances,	<b>This happens:</b> I see an option to transfer funds.
<b>As</b> Carol,	When I select the transfer option,	I choose between which accounts I want to transfer funds.
<b>I want to</b> transfer funds between accounts	When I select the "transfer from" option,	I see a list of my available accounts and balances.
<b>so that</b> each account has the correct amount of funds	When I select the "transfer to" option,	I see a list of my available accounts and balances.
Value	Jennifer	Estimate

[FIGURE 15-3](#): A user story and acceptance criteria.

Even if you don't describe your requirements in a user story format, consider adding validation steps to each of your requirements. Acceptance criteria don't just help the product owner review requirements; they help the development team understand how to create the product in the first place.

### **Face-to-face communication**

Have you ever had a conversation with someone and known, just by looking at that person's face, that he or she didn't understand you? In [Chapter 14](#), we explain that face-to-face conversations are the quickest, most effective form of communication. This is because humans convey information with more than just words; our facial expressions, gestures, body language, and even where we are looking contribute to communicating and understanding one another.

Face-to-face communication helps ensure quality on agile projects because it

leads to better interpretation of requirements, roadblocks, and discussions between scrum team members. Regular face-to-face communication requires a collocated scrum team.

## **Sustainable development**

Chances are, at some point in your life, you've found yourself working or studying long hours for an extended period of time. You may have even pulled an all-nighter or two, getting no sleep at all for a night. How did you feel during this time? Did you make good decisions? Did you make any silly mistakes?

Unfortunately, many teams on traditional projects find themselves working long, crazy hours, especially toward the end of a project, when a deadline is looming and it seems like the only way to finish is to spend weeks working extra-long days. Those long days often mean more problems later, as team members start making mistakes — some silly, some more serious — and eventually burn out.

On agile projects, scrum teams help ensure that they do quality work by creating an environment where members of the development team sustain a constant working pace throughout the project. Working in sprints helps sustain a constant working pace; when the development team chooses the work it can accomplish in each sprint, it shouldn't have to rush at the end.

The development team can determine what sustainable means for itself, whether that means working a regular 40-hour workweek, a schedule with more or fewer days or hours, or working outside a standard nine-to-five time frame.



TIP If your fellow scrum team members start coming to work with their shirts on inside out, you might want to double-check that you're maintaining a sustainable development environment.

Keeping the development team happy, rested, and able to have a life outside of work can lead to fewer mistakes, more creativity and innovation, and better overall products.

Being proactive about quality saves you a lot of headaches in the long run. It

is much easier and more enjoyable to work on a product with fewer defects to fix. The next section discusses an agile approach that addresses quality from both a proactive and a reactive standpoint: inspect and adapt.

## ***Quality through regular inspecting and adapting***

The agile tenet of inspect and adapt is a key to creating quality products. Throughout an agile project, you look at both your product and your process (inspect) and make changes as necessary (adapt). [Chapters 7](#) and [10](#) have more information about this tenet.

In the sprint review and sprint retrospective meetings, agile project teams regularly step back and review their work and methods and determine how to make adjustments for a better project. We provide details on the sprint review and sprint retrospective in [Chapter 10](#). Following is a quick overview of how these meetings help ensure quality on agile projects.

In a sprint review, agile project teams review requirements completed at the end of each sprint. Sprint reviews address quality by letting project stakeholders see working requirements and provide feedback on those requirements throughout the course of the project. If a requirement doesn't meet stakeholder expectations, the stakeholders tell the scrum team immediately. The scrum team can then adjust the product in a future sprint. The scrum team can also apply its revised understanding of how the product needs to work on other product requirements.

In a sprint retrospective, scrum teams meet to discuss what worked and what might need adjusting at the end of each sprint. Sprint retrospectives help ensure quality by allowing the scrum team to discuss and immediately fix problems. Sprint retrospectives also allow the team to come together and formally discuss changes to the product, project, or work environment that might increase quality.

The sprint review and sprint retrospective aren't the only opportunities for inspecting and adapting for quality on an agile project. Agile approaches encourage reviewing work and adjusting behavior and methods throughout each workday. Daily inspecting and adapting everything you do on the project help ensure quality.

Another way to manage and help assure quality on an agile project is to use automated testing tools. The next section explains why automated testing is

important to agile projects and how to incorporate automated testing into your project.

## **Automated testing**

Automated testing is the use of software to test your product. Automated testing is critical to agile projects. If you want to quickly create software functionality that meets the definition of done — coded, tested, integrated, and documented — you need a way to quickly test each piece of functionality as it's created. Automated testing means quick and robust testing on a daily basis. Agile teams continually increase the frequency with which they automatically test their system so they can continually decrease the time it takes them to complete and deploy new valuable functionality to their customers.



**TIP** Project teams won't become agile without automated testing. Manual testing simply takes too long.

Throughout this book, we explain how agile project teams embrace low-tech solutions. Why, then, is there a section in this book about automated testing, a rather high-tech quality management technique? The answer to this question is efficiency. Automated testing is like the spell-check feature in word-processing programs. As a matter of fact, spell-checking is a form of automated testing. In the same way, automated testing is a much quicker and often more accurate — thus, more efficient — method of finding software defects than manual testing.

To develop a product using automated testing, development teams develop and test using the following steps:

- 1. Develop code and automated tests in support of user stories during the day.**
- 2. Create an integrated code build at the end of each day.**
- 3. Schedule the automated testing software to test the newest build overnight.**
- 4. Check the automated test results first thing each morning.**

## 5. Fix any defects immediately.



TIP

Comprehensive code testing while you're sleeping is cool.

Automated testing allows development teams to take advantage of non-working time for productivity and to have rapid create-test-fix cycles. Also, automated testing software can often test requirements quicker and with more accuracy and consistency than a person testing those requirements.

Today's market has a lot of automated testing tools. Some automated testing tools are open-source and free; others are available for purchase. The development team needs to review automated testing options and choose the tool that will work best.

Automated testing changes the work for people in quality roles on the development team. Traditionally, a large part of a quality management person's work involved manually testing products. The tester on a traditional project would use the product and look for problems. With automated testing, however, quality activities largely involve creating tests to run on automated testing software. Automated testing tools augment, rather than replace, people's skills, knowledge, and work.



WARNING It is still a good idea to have humans periodically check that the requirements you're developing work correctly, especially when you first start using an automated testing tool. Any automated tool can have hiccups from time to time. By manually double-checking (sometimes called smoke-testing) small parts of automated tests, you help avoid getting to the end of a sprint and finding out that your product doesn't work like it should.

You can automate almost any type of software test. If you're new to software development, you may not know that there are many different types of software testing. A small sample includes the following:

» **Unit testing:** Tests individual units, or the smallest parts, of product code.

- » **Regression testing:** Tests an entire product from start to finish, including requirements you have tested previously.
- » **User acceptance testing:** Product stakeholders or even some of the product's end users review a product and accept it as complete.
- » **Functional testing:** Tests to make sure the product works according to acceptance criteria from the user story.
- » **Integration testing:** Tests to make sure the product works with other parts of the product.
- » **Enterprise testing:** Tests to make sure the product works with other products in the organization, as necessary.
- » **Performance testing:** Tests how fast a product runs on a given system under different scenarios.
- » **Load testing:** Tests how well a product handles different amounts of concurrent activity.
- » **Smoke testing:** Tests on small but critical parts of code or of a system to help determine if the system as a whole is likely to work.
- » **Static testing:** Focuses on checking code standards, rather than working software.

Automated testing works for these tests and the many other types of software tests out there.

As you may understand by now, quality is an integral part of agile projects. Quality is just one factor, however, that differentiates risk on agile projects from traditional projects. In the next sections, you see how risk on traditional projects compares to risk on agile projects.

## *What's Different about Agile Risk Management?*

Risk refers to the factors that contribute to a project's success or failure. On agile projects, risk management doesn't have to involve formal risk documentation and meetings. Instead, risk management is built into scrum roles, artifacts, and events. In addition, consider the following agile principles

that support risk management:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Working software is the primary measure of progress.

The preceding principles, and any practice that demonstrates those principles, significantly mitigate or eliminate many risks that frequently lead to project challenges and failure.

According to the Standish Group's "2015 Chaos Report," a study of 10,000 software projects, small agile projects are 30 percent more likely to succeed than traditional projects. See [Figure 15-4](#). Medium-sized projects are four times (400 percent) more likely to succeed with an agile approach than a traditional approach, and large, complex projects are six times (600 percent) more likely to succeed with an agile approach.

## CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

The resolution of all software projects from FY2011-2015 within the new CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000.

**FIGURE 15-4:** Standish Group's "2015 Chaos Report."

Table 15-2 shows some of the differences between risk on traditional projects and on agile projects.

## **TABLE 15-2 Traditional versus Agile Risk**

<i>Risk Management with Traditional Approaches</i>	<i>Risk Dynamics with Agile Approaches</i>
Large numbers of projects fail or are challenged.	Risk of catastrophic failure — spending large amounts of money with nothing to show — is almost eliminated.
The bigger, longer, and more complex the project, the more risky it is. Risk is highest at the end of a project.	Product value is gained immediately, rather than sinking costs into a project for months or even years with the growing chance of failure.
Conducting all the testing at the end of a project means that finding serious problems can put the entire project at risk.	Testing occurs while you develop. If a technical approach, a requirement, or even an entire product is not feasible, the development team discovers this in a short time, and you have more time to course correct. If correction is not possible, stakeholders spend less money on a failed project.

Projects are unable to accommodate new requirements mid-project without increased time and cost because extensive sunk cost exists in even the lowest-priority requirements.

Change for the benefit of the product is welcomed. Agile projects accommodate new high-priority requirements without increasing time or cost by removing a low-priority requirement of equal time and cost.

Traditional projects require time and cost estimates at the project start, when teams know the least about the project. Estimates are often inaccurate, creating a gap between expected and actual project schedules and budgets.

Project time and cost is estimated using the scrum team's actual performance, or velocity. You refine estimates throughout the project, because the longer you work on a project, the more you learn about the project, the requirements, and the scrum team.

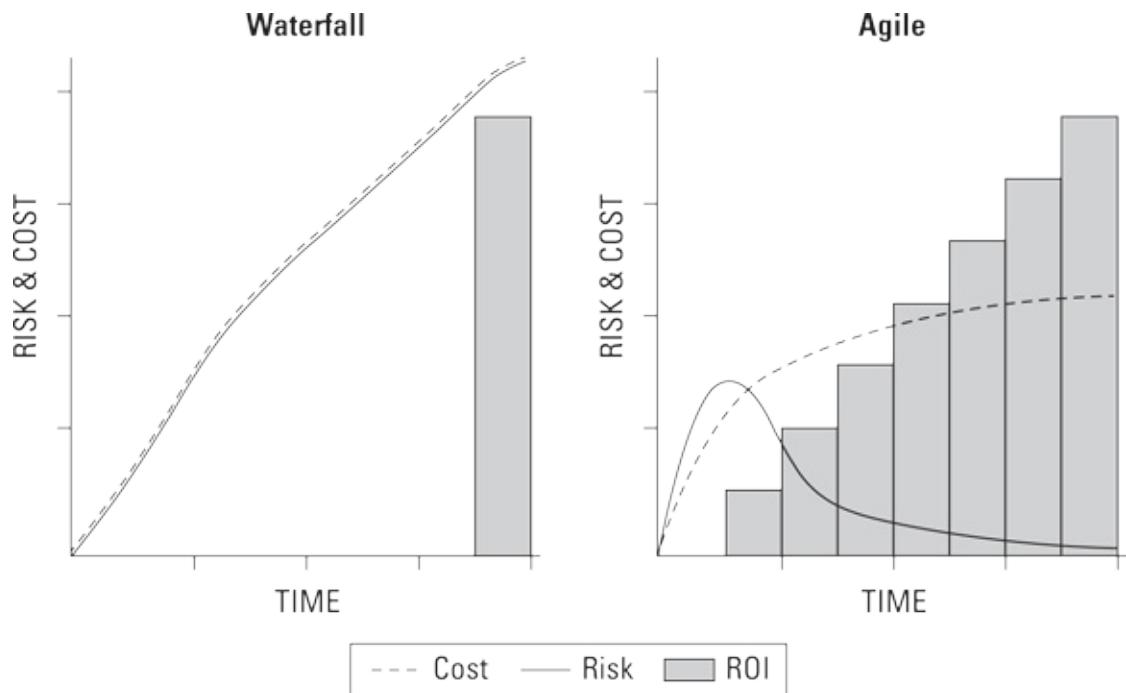
When stakeholders don't have a unified goal, they can end up confusing the project team with conflicting information about what the product should achieve.

A single product owner is responsible for creating a vision for the product and represents the stakeholders to the project team.

Unresponsive or absent stakeholders can cause project delays and result in products that do not achieve the right goals.

The product owner is responsible for providing information about the product immediately. In addition, the scrum master helps remove impediments on a daily basis.

Risk on agile projects declines as the project progresses. [Figure 15-5](#) shows a comparison of risk and time between waterfall projects and agile projects.



**FIGURE 15-5:** Agile projects' declining risk model.

All projects have some risk, regardless of your project approach. However, with agile project management, the days of catastrophic project failure — spending large amounts of time and money with no return on investment (ROI) — are over. The elimination of large-scale failure is the biggest

difference between risk on traditional projects and on agile projects. In the next section, you see why.

## ***Managing Agile Risk***

In this section, you examine key structures of agile projects that reduce risk over the life of the project. You find out how to use agile tools and events to find risks at the right time in a project and how to prioritize and mitigate those risks.

### ***Reducing risk inherently***

Agile approaches, when implemented correctly, inherently reduce risk in product development. Developing in sprints ensures a short time between project investment and proof that the product works. Sprints also provide the potential for a project to generate revenue early. The sprint review, the sprint retrospective, and the product owner's involvement during each sprint provide constant product feedback to the development team. Ongoing feedback helps prevent deviations between product expectations and the completed product.

Three especially important factors in risk reduction on agile projects are the definition of done, self-funding projects, and the idea of failing fast. You find out more about each of these factors in this section.

### ***Risk and the definition of done***

In [Chapter 10](#), we discuss when a requirement is done. To consider a requirement complete and ready to demonstrate at the end of a sprint, that requirement must meet the scrum team's definition of done. The product owner and the development team agree upon the details of the definition; definitions of done usually include the following:

- » **Developed:** The development team must fully create the working product requirement.
- » **Tested:** The development team must have tested that the product works correctly and is defect-free.
- » **Integrated:** The development team must have ensured that the requirement works with the whole product and any related systems.

- » **Documented:** The development team must have created notes about how it created the product and the rationale behind key technical decisions made.

[Figure 15-6](#) shows a sample definition of done, with details.

## DEFINITION OF DONE

SPRINT	RELEASE	RISKS ACCEPTED
QA	Staging	
Unit/Dev	Perf	Mark
Functional	Load	Mike
Integration	Security	Sarah
Regression	Focus Groups	Jim
User Accept	Enterprise	Deepa
Static	Regulatory	
Peer Review	User Docs	
→ xDocs	Training	
→ Wikis		

[FIGURE 15-6:](#) Sample definition of done.

The product owner and the development team may also create a list of acceptable risks. For example, they may agree that end-to-end regression testing or performance testing is overkill for the sprint definition of done. Or, with cloud computing, load testing may not be as crucial because additional

capacity can be easily and quickly added on demand at nominal costs. Acceptable risks allow the development team to concentrate on the most important activities.

The definition of done drastically changes the risk factor for agile projects. By creating a product that meets the definition of done in every sprint, you end each sprint with a working build and usable functionality. Even if outside factors cause a project to end early, project stakeholders will always see some value and have working functionality to use now and build upon later.

### ***Self-funding projects***

Agile projects can mitigate financial risk in a unique way that traditional projects cannot: the self-funding project. [Chapter 13](#) includes examples of self-funding projects. If your product is an income-generating product, you could use that income to help fund the rest of your project.

In [Chapter 13](#), we show you two different project ROI models. Here they are again, in [Tables 15-3](#) and [15-4](#). The projects in both tables create identical products.

**TABLE 15-3 Income from a Traditional Project with a Final Release after Six Months**

<i>Month</i>	<i>Income Generated</i>	<i>Total Project Income</i>
January	\$0	\$0
February	\$0	\$0
March	\$0	\$0
April	\$0	\$0
May	\$0	\$0
June	\$0	\$0
July	\$100,000	\$100,000

**TABLE 15-4 Income from an Agile Project with Monthly Releases and a Final Release after Six Months**

<i>Month/Release</i>	<i>Income Generated</i>	<i>Total Project Income</i>
January	\$0	\$0
February	\$15,000	\$15,000
March	\$15,000	\$30,000
April	\$15,000	\$45,000
May	\$15,000	\$60,000
June	\$15,000	\$75,000
July	\$100,000	\$100,000

March	\$25,000	\$40,000
April	\$40,000	\$80,000
May	\$70,000	\$150,000
June	\$80,000	\$230,000
July	\$100,000	\$330,000

In [Table 15-3](#), the project created \$100,000 in income after six months of development. Now compare the ROI in [Table 15-3](#) to the ROI in [Table 15-4](#).

In [Table 15-4](#), the project generated income with the very first release. By the end of six months, the project had generated \$330,000 — \$230,000 more than the project in [Table 15-3](#).

The capability to generate income in a short amount of time has a number of benefits for companies and project teams. Self-funding agile projects make good financial sense for almost any organization, but they can be especially useful to organizations that may not have the funds to create a product upfront. For groups short on cash, self-funding can enable projects that would otherwise not be feasible.

Self-funding projects also help mitigate the risk that a project will be cancelled due to lack of funds. A company emergency may dictate diverting a traditional project's budget elsewhere, delaying or cancelling the project. However, an agile project that generates additional revenue with every release has a good chance of continuing during a crisis.

Finally, self-funding projects help sell stakeholders on a project in the first place; it's hard to argue with a project that provides continuous value and pays for at least part of the project costs from the start.

## ***Failing fast***

All product development efforts carry some risk of failure. Testing within sprints introduces the idea of *failing fast*: Instead of sinking costs into a long effort for requirements, design, and development, and then finding problems that will prevent the project from moving forward during the testing phase, development teams on agile projects can identify critical problems within a few sprints. This quantitative risk mitigation can save organizations large amounts of money.

[Tables 15-5](#) and [15-6](#) illustrate the difference in sunk costs for a failed

waterfall project and a failed agile project. The projects in both tables are for identical products with identical costs.

**TABLE 15-5 Cost of Failure on a Waterfall Project**

<i>Month</i>	<i>Phase and Issues</i>	<i>Sunk Project Cost</i>	<i>Total Sunk Project Cost</i>
January	Requirements Phase	\$80,000	\$80,000
February	Requirements Phase	\$80,000	\$160,000
March	Design Phase	\$80,000	\$240,000
April	Design Phase	\$80,000	\$320,000
May	Design Phase	\$80,000	\$400,000
June	Development Phase	\$80,000	\$480,000
July	Development Phase	\$80,000	\$560,000
August	Development Phase	\$80,000	\$640,000
September	Development Phase	\$80,000	\$720,000
October	QA Phase: Large-scale problem uncovered during testing.	\$80,000	\$800,000
November	QA Phase: Development team attempted to resolve problem to continue development.	\$80,000	\$880,000
December	Project cancelled; product not viable.	0	\$880,000

**TABLE 15-6 Cost of Failure on an Agile Project**

<i>Month</i>	<i>Sprint and Issues</i>	<i>Sunk Project Cost</i>	<i>Total Sunk Project Cost</i>
January	Sprint 1: No issues. Sprint 2: No issues.	\$80,000	\$80,000
February	Sprint 3: Large-scale problem uncovered during testing resulted in failed sprint. Sprint 4: Development team attempted to resolve problem to continue development; sprint ultimately failed.	\$80,000	\$160,000
Final	Project cancelled; product not viable.	0	\$160,000

In [Table 15-5](#), the project stakeholders spent 11 months and close to a million dollars to find out that a product idea would not work. Compare the sunk cost in [Table 15-5](#) to that in [Table 15-6](#).

By testing early, the development team from [Table 15-6](#) determined that the

product would not work by the end of February, spending less than one-sixth of the time and money spent in the project in [Table 15-5](#).



**REMEMBER** Because of the definition of done, even failed projects produce something tangible that an organization may leverage or improve. For example, the failed project in [Table 15-5](#) would have provided working functionality in the first two sprints.

The concept of failing fast can apply beyond technical problems with a product. You can also use development within sprints and fast failure to see if a product will work in the marketplace, and to cancel the project early if it looks like customers won't buy or use the product. By releasing small parts of the product and testing the product with potential customers early in the project, you get a good idea of whether your product is commercially viable, and save large amounts of money if you find that people will not buy the product. You also discover important changes you might make to the product to better meet customer needs.

Finally, failing fast does not necessarily mean project cancellation. If you find catastrophic issues when sunk costs are low, you may have the time and budget to determine a completely different approach to create a product.

The definition of done, self-funding projects, and the idea of failing fast, along with the foundation of agile principles, all help lower risk on agile projects. In the next section, you see how to actively use agile project management tools to manage risk.

## ***Identifying, prioritizing, and responding to risks early***

Although the structure of agile projects inherently reduces many traditional risks, development teams still should be aware of the problems that can arise during a project. Scrum teams are self-managing; in the same way that they are responsible for quality, scrum teams are responsible for trying to identify risks and ways to prevent those risks from materializing.



**TIP** On agile projects, you prioritize the highest-value and highest-risk

requirements first.

Instead of spending hours or days documenting all of a project's potential risks, the likelihood of those risks happening, the severity of those risks, and ways to mitigate those risks, scrum teams use existing agile artifacts and meetings to manage risk. Scrum teams also wait until the last responsible minute to address risk, when they know the most about the project and problems that are more likely to arise. [Table 15-7](#) shows how scrum teams can use the different agile project management tools to manage risk at the right time.

## **TABLE 15-7 Agile Project Risk Management Tools**

<b><i>Artifact or Meeting</i></b>	<b><i>Role in Risk Management</i></b>
Product vision	The product vision statement helps unify the project team's definition of product goals, mitigating the risk of misunderstandings about what the product will need to accomplish. While creating the product vision, the project team might think of risks on a very high level, based on marketplace and customer feedback, and inline with organizational strategy. Find out more about the product vision in <a href="#">Chapter 7</a> .
Product roadmap	The product roadmap provides a visual overview of the project's requirements and priorities. This overview allows the project team to quickly identify gaps in requirements and incorrectly prioritized requirements. Find out more about the product roadmap in <a href="#">Chapter 7</a> .
Product backlog	The product backlog is a tool for accommodating change in the project. Being able to add changes to the product backlog and reprioritize requirements regularly helps turn the traditional risk associated with scope changes into a way to create a better product. Keeping the requirements and the priorities in the product backlog current helps ensure that the development team can work on the most important requirements at the right time. Find out more about the product backlog in <a href="#">Chapters 7 and 8</a> .
Release planning	During release planning, the scrum team discusses risks to the release and how to mitigate those risks. Risk discussions in the release planning meeting should be high-level and relate to the release as a whole. Address risks with individual requirements in the sprint planning meetings. Find out more about release planning in <a href="#">Chapter 8</a> .
Sprint planning	During each sprint-planning meeting, the scrum team discusses risks to the specific requirements and tasks in the sprint and how to mitigate those risks. Risk discussions during sprint planning can be done in depth, but should only relate to the current sprint. Find out more about sprint planning in <a href="#">Chapter 8</a> .
Sprint backlog	The burndown chart on the sprint backlog provides a quick view of the sprint status. This quick view helps the scrum team manage risks to the sprint as they arise and minimize their effect by addressing problems immediately. Find out more about sprint backlogs and how burndown charts show project status in <a href="#">Chapter 9</a> .
Daily scrum	During each daily scrum, development team members discuss roadblocks. Roadblocks, or impediments, are sometimes risks. Talking about roadblocks every day gives the development team and the scrum master the chance to mitigate those risks immediately. Find out more about the daily scrum in <a href="#">Chapter 9</a> .
Task board	The task board provides an unavoidable view of the sprint status, allowing the scrum team to catch risks to the sprint and manage them right away.

Find out more about task boards in [Chapter 9](#).

---

Sprint review	During the sprint review, the scrum team regularly ensures that the product meets stakeholders' expectations. The sprint review also provides opportunities for stakeholders to discuss changes to the product to accommodate changing business needs. Both aspects of the sprint review help manage the risk of getting to the end of a project with the wrong product. Find out more about sprint reviews in <a href="#">Chapter 10</a> .
Sprint retrospective	During the sprint retrospective, the scrum team discusses issues with the past sprint and identifies which of those issues may be risks in future sprints. The development team needs to determine ways to prevent those risks from becoming problems again. Find out more about sprint retrospectives in <a href="#">Chapter 10</a> .

---

---

The artifacts and meetings discussed in this section systematically help agile teams manage risk on an agile project by addressing risk by the responsible roles at appropriate times. The larger and more complex the project, the higher the likelihood that an agile approach can eliminate the risk of failure.

## Part 5

# Ensuring Agile Success

## **IN THIS PART ...**

Build a foundation through organizational and individual commitment to becoming more agile.

Choose a project and create an environment that will optimize agile transition success.

Scale agile techniques across multi-team projects appropriately.

Become a change agent in your organization and help avoid common pitfalls in agile transitions.

# Chapter 16

## Building a Foundation

---

### IN THIS CHAPTER

- » Obtaining organizational and individual commitment
- » Assembling teams with the necessary skills and abilities
- » Establishing an appropriate environment
- » Investing in training
- » Securing initial and ongoing support

To successfully move from traditional project management processes to agile processes, you must start with a good foundation. You need commitment, both from your organization and from people as individuals, and you need to find a good project team for your first agile project, providing them an environment conducive to agile approaches. You want to find the right training for your project team, and sustainably support your organization's agile approach so that it can grow beyond your first project.

In this chapter, we show you how to build a strong agile foundation within your organization.

### *Organizational and Individual Commitment*

Commitment to agile project management means making an active, conscious effort to work with new methods and to abandon old habits. Commitment at both an individual level and at an organizational level is critical to agile transition success.

Without organizational support, even the most enthusiastic agile project team members may find themselves forced back into old project management processes. Without the commitment of individual project team members, a

company that embraces agile approaches may encounter too much resistance, or even sabotage, to be able to become an agile organization.

The following sections provide details on how organizations and people can support an agile transition.

## ***Organizational commitment***

Organizational commitment plays a large role in agile transition. When a company and the groups in that company embrace agile principles, the transition can be easier for the project team members.

Organizations can commit to an agile transition by doing the following:

- » Engaging an experienced agile expert to create a realistic transition plan and to guide the company through that plan
- » Investing in employee training, starting with the members of the company's first agile project team and the leadership at all levels who support them
- » Allowing scrum teams to abandon waterfall processes, meetings, and documents in favor of streamlined agile approaches
- » Ensuring all scrum team members necessary for each agile project are dedicated: an empowered product owner, a cross-functional development team of multi-skilled people, and an influential servant-leader scrum master
- » Encouraging development teams to continuously increase their skill sets
- » Providing automated testing tools and a continuous integration framework
- » Logistically supporting scrum team collocation
- » Allowing scrum teams to manage themselves
- » Giving the agile project team the time and freedom to go through a healthy trial-and-error process
- » Revising employee performance reviews to emphasize team performance
- » Encouraging agile project teams and celebrating successes

Organizational support is also important beyond the agile transition. Companies can ensure that agile processes continue to work by hiring with

agile project teams in mind and by providing agile training to new employees. Organizations can also engage the ongoing support of an agile mentor, who can guide project teams as they encounter new and challenging situations.

Organizations, of course, are made up of individuals. Organizational commitment and individual commitment go hand in hand.

## ***Individual commitment***

Individual commitment has an equal role to organizational commitment in agile transitions. When each person on a project team works at adopting agile practices, the changes become easier for everyone on the project team.

People can individually commit to an agile transition by using these methods:

- » Attending training and conferences and being willing to learn about agile methods
- » Being open to change, willing to try new processes, and making an effort to adapt new habits
- » Resisting the temptation to fall back on old processes
- » Acting as a peer coach for project team members who are less experienced in agile techniques
- » Allowing themselves to make mistakes and learn from those mistakes
- » Reflecting on each sprint honestly in the sprint retrospective and committing to improvement efforts
- » Actively becoming multi-skilled development team members
- » Letting go of ego and working as a part of a team
- » Taking responsibility for successes and failures as a team
- » Taking the initiative to be self-managing
- » Being active and present throughout each agile project

Like organizational commitment, individual commitment is important beyond the agile transition period. The people on the first agile project team will become change agents throughout the company, setting the stage and exemplifying for other project teams how to effectively work with agile methods.

## **Getting commitment**

Commitment to agile methods may not be instant. You'll need to help people in your organization overcome the natural impulse to resist change.

A good early step in an agile transition is to find an *agile champion*, a senior-level manager or executive who can help ensure organizational change. The fundamental process changes that accompany agile transitions require support from the people who make and enforce business decisions. A good agile champion will be able to rally the organization and its people around process changes.

Another important way to get commitment is to identify challenges with the organization's current projects and provide potential solutions with agile approaches. Agile project management can address many problems, including issues with product quality, customer satisfaction, team morale, budget and schedule overruns, funding, portfolio management, and overall project issues.

Finally, highlight some of agile project management's overall benefits. Some of the real and tangible benefits that drive shifts from traditional methods of project management to agile methods include the following:

- » **Happier customers:** Agile projects often have higher customer satisfaction because agile project teams produce working products quickly, can respond to change, and collaborate with customers as partners.
- » **Profit benefits:** Agile approaches allow project teams to deliver functionality to market quicker than with traditional approaches. Agile organizations can realize higher return on investment, often resulting in self-funded projects.
- » **Defect reduction:** Quality is a key part of agile approaches. Proactive quality measures, continuous integration and testing, and continuous improvement all contribute to higher-quality products.
- » **Improved morale:** Agile practices such as sustainable development and self-managing development teams can mean happier employees, improved efficiency, and less company turnover.

You can find more benefits of agile project management in [Chapter 19](#).

## **Can you make the transition?**

You've established many valuable reasons for moving to an agile approach, and your case looks good. But will your organization be able to make the transition? Here are some key questions to consider:

- » **What are the organizational roadblocks?** Does your organization have a value-delivery culture or a risk-management culture? Does it support coaching and mentorship alongside management? Is there support for training? How does the organization define success? Does it have an open culture that will embrace a high visibility of project progress?
- » **How are you doing business today?** How are projects planned at the macro level? Is the organization fixated on fixed scope? How engaged are business representatives? Do you outsource development?
- » **How do your teams work today, and what will need to shift under agile methods?** How ingrained is waterfall? Does the team have a strong command-and-control mentality? Can good ideas come from anywhere? Is there trust in the team? Are people shared across teams? What do you need to ask for to secure a shift? Can you get people, tools, space, and commitment to pilot the change?
- » **What are the regulatory challenges?** Are there processes and procedures that relate to regulatory requirements? Are these requirements imposed upon you from externally or internally adopted regulations and standards? Will you need to create additional documentation to satisfy regulatory requirements? Are you likely to be audited for compliance, and what would be the cost of noncompliance?

As you review your analysis of the roadblocks and challenges, you may uncover the following concerns:

- » **Agile approaches reveal that the organization needs to change.** As you compare agile practices and results with what you have done traditionally, you may reveal that performance has not been all it could have been. You need to tackle this head on. Your organization has been operating within a framework of how projects were expected to be run. Your organization has done its best to produce a result, often in the face of extreme challenges. For all parties involved, you have to acknowledge their efforts

and introduce the potential of agile processes to allow them to produce yet greater results.

- » **Project management leaders may misinterpret agile techniques as insufficient.** Often the values and principles of the Agile Manifesto are misinterpreted to mean agile frameworks involve insufficient planning and documentation and attempt to disregard generally accepted project management standards. Experienced project managers may view some of that value slipping away in a transition to agile processes. Take every opportunity to clarify what agile values and principles support and do not support. Show how each principle addresses the same challenges that traditional project management attempts to resolve, and how agile techniques are an extension of project managers' capabilities and career, not as devaluing anything they've worked hard to secure.
- » **Moving from a leadership to a service model can be challenging.** Agile leaders are service oriented. Command and control gives way to facilitation. Servant leadership is a big shift for many project teams and functional managers. Demonstrate how the shift provides more effective outcomes for everyone. You can read more about servant leadership in [Chapter 14](#).

Keep in mind that some resistance will arise; change can't happen without opposition. Be ready for resistance, but don't let it thwart your overall plan.

## ***Timing the transition***

Organizationally, you can start your initiative to move to an agile approach at any time. You might consider a few optimal times:

- » **When you need to prove that agile project management is necessary:** Use the end of a large project, when you can see clearly what did not work (for example, during a sunset review). You'll be able to demonstrate clearly the issues with waterfall, and you'll gain a springboard for piloting your first agile project.
- » **When your challenge is doing accurate budgeting:** Run your first agile project in the quarter before the start of the annual budget year (namely, one quarter before the end of the current budget cycle). You'll get metrics from your first project that will allow you to be more informed when

planning next year's budget.

- » **When you're starting a new project:** Moving to agile processes when you have a new project lets you start fresh without the baggage of old approaches.
- » **When you're trying to reach a new market or industry:** Agile techniques allow you to deliver quick innovation to help your organization create products for new types of customers.
- » **When you have new leadership:** Management changes are great opportunities for setting new expectations with agile approaches.

Although you can take advantage of any of these opportunities to start using agile processes, they're not required. The best time to become more agile is ... today!

## *Choosing the Right Pilot Team Members*

Determining the right people to work with, especially in the early stages, is important to agile project success. Here are things to think about when choosing people for the different roles in your organization's first agile project.

### *The agile champion*

At the beginning of an agile transition, the agile champion will be a key person in helping ensure the project team can succeed. This person should be able to effectively and quickly influence each level of the organization that affects the pilot agile teams' chances for success. A good agile champion should be able to do all these tasks:

- » Be passionate about agility and the organizational and market issues agile approaches will address.
- » Make decisions about company processes. If there is a status quo, the agile champion should be able to influence a change.
- » Get the organization excited about what's possible with agile processes.

- » Regularly and directly collaborate with and support the project team as it goes through the steps to establish agile processes.
- » Acquire the project team members necessary for success, both for the first project and in the long term.
- » Be an escalation point to remove unnecessary distractions and non-agile processes.

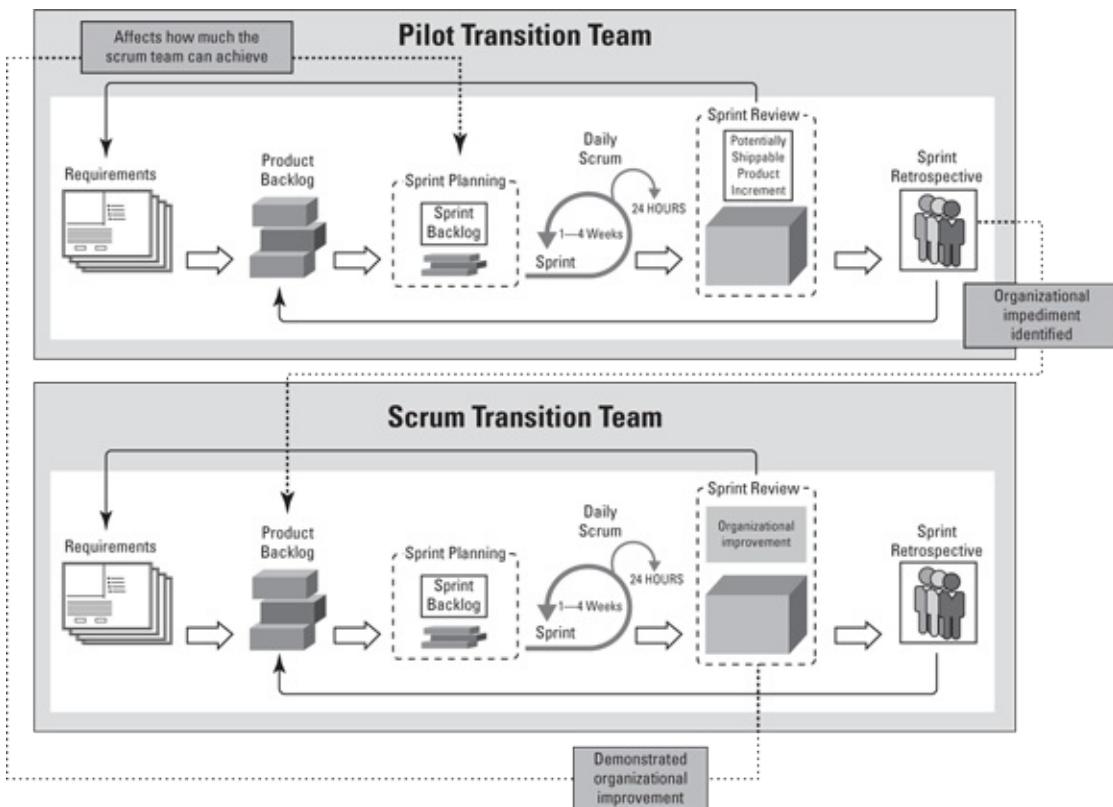
When choosing an agile champion, look for someone who has authority in the organization — whose voice is respected and who has led change initiatives successfully in the past.

## ***The agile transition team***

As important as the agile champion is, one person can't do everything. The agile champion should work together with other organizational leaders whom the agile project team relies on for support in the transition. Together, the agile transition team removes organizational impediments to ensure the success of the pilot team and future agile teams. The agile transition team should

- » Be committed to organizational success through the continuous support of pilot agile teams.
- » Establish a clear vision and roadmap for how the organization will become agile.
- » Be organized like a scrum team, with a product owner (agile champion), development team (leaders who can make organizational changes in support of the pilot scrum teams), and a scrum master (an organizational leader who can focus on helping the agile transition team adopt agile principles and enforce the rules of scrum).
- » Operate as a scrum team, holding all five scrum events and implementing all three scrum artifacts.

[Figure 16-1](#) illustrates how the agile transition team's and pilot scrum team's sprint cadences are aligned. Impediments identified in the sprint retrospective of the pilot team become backlog items for the transition team to resolve as process improvements for the pilot team.



**FIGURE 16-1:** Alignment of the agile transition team and the pilot scrum team cadences.

Not only does the agile transition team provide systematic support for the pilot scrum team, but the organizational leadership also becomes more agile by using scrum alongside the pilot team.

## *The product owner*

With an agile champion and an agile transition team in place, the focus turns to pilot scrum teams. The pilot scrum team product owners should come from the business side of the organization, aligning the business with technology. During the first agile project, the product owner may need to acclimate to working on the project daily with the development team. A good product owner should

- » Be decisive.
- » Be an expert about customer requirements and business needs.
- » Have the business authority and be empowered to prioritize and reprioritize product requirements.
- » Be organized enough to manage ongoing changes to the product backlog.

- » Be committed to working with the rest of the scrum team and to being available to the development team daily throughout a project.
- » Have the ability to obtain project funding and other resources.

When choosing a product owner for your first agile project, find someone who can provide product expertise and commitment to the project.

## ***The development team***

On agile projects, the self-managing development team is central to the success of the project. The development team determines how to go about the work of creating the product. Good development team members should be able to do the following:

- » Be versatile.
- » Be willing to work cross-functionally.
- » Plan a sprint and self-manage around that plan.
- » Understand the product requirements and provide effort estimates.
- » Provide technical advice to the product owner so that he or she can understand the complexity of the requirements and make appropriate decisions.
- » Respond to circumstances and adjust processes, standards, and tools to optimize performance.

Intellectually curious developers, eager to learn new things and contribute to project goals in a variety of ways, are most likely to thrive in an agile environment. When choosing a development team for the pilot project, select people who are open to change, enjoy a challenge, like to be in the forefront of new developments, and are willing to do whatever it will take to ensure success, including learning and using new skills outside their existing skill set.

## ***The scrum master***

The scrum master on a company's first agile project may need to be more sensitive to potential development team distractions than on later projects. A good scrum master should

- » Have influence (clout).
- » Have enough organizational influence to remove outside distractions that prevent the project team from successfully using agile methods.
- » Know enough about agile project management to be able to help the project team uphold agile processes throughout a project.
- » Have the communication and facilitation skills to guide the development team in reaching consensus.
- » Trust enough to step back and allow the development team to organize and manage itself.

When determining the scrum master for a company's first agile project, you want to select someone who is willing to be a servant-leader. At the same time, the scrum master will need to have a strong enough temperament to help thwart distractions and uphold agile processes in the face of organizational and individual resistance.

## ***The project stakeholders***

On an organization's first agile project, good project stakeholders should

- » Be involved.
- » Defer to the product owner for final product decisions.
- » Attend sprint reviews and provide product feedback.
- » Understand agile processes. Sending project stakeholders to the same training as the rest of the project team will help them be more comfortable with new processes.
- » Receive project information in agile formats, such as sprint reviews, product backlogs, and sprint backlogs.
- » Provide details when the product owner and development team have questions.
- » Work collaboratively with the product owner and the rest of the project team.

The project stakeholders for your agile project should be trustworthy, cooperative, and active contributors to a project.

## ***The agile mentor***

An agile mentor, sometimes called an agile coach, is key to keeping teams and organizations on track while learning scrum and beginning to establish a more agile environment. A good agile mentor should

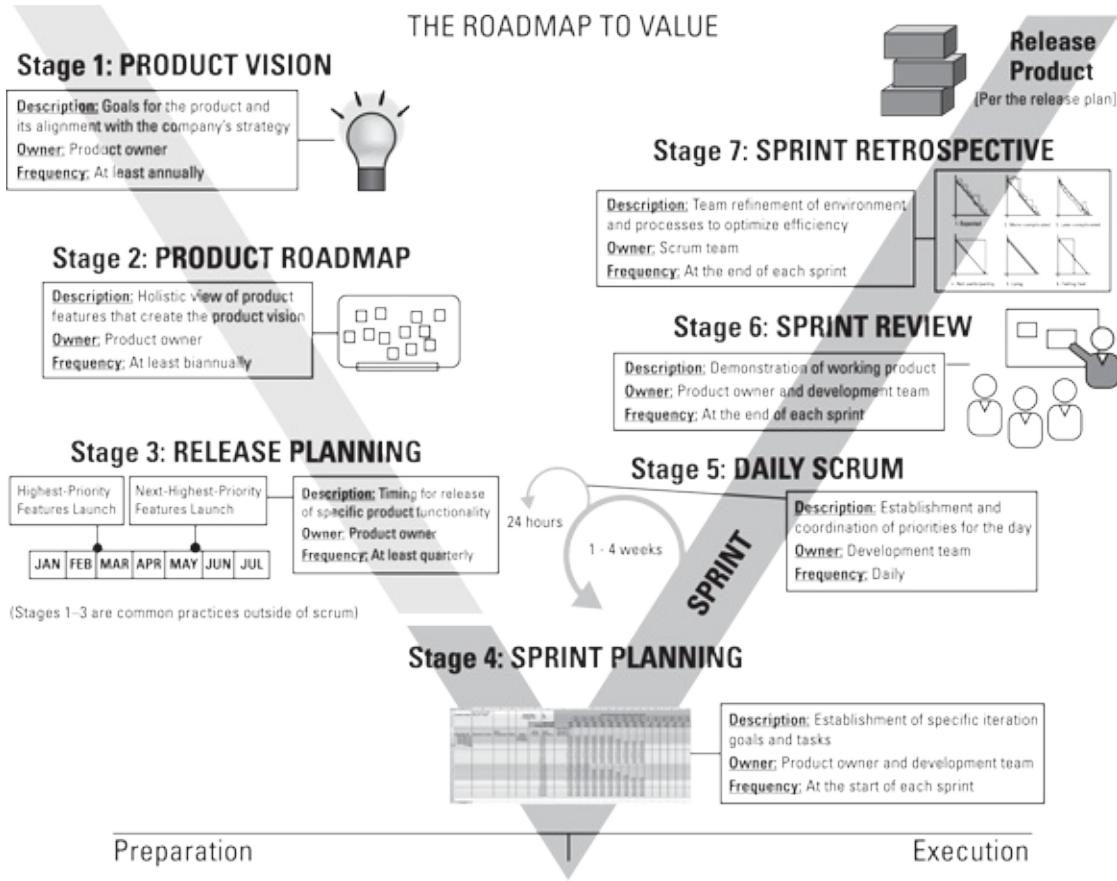
- » Be experienced.
- » Be an expert at agile processes, especially in the agile processes your organization chooses.
- » Be familiar with projects of different sizes, large and small.
- » Help teams self-manage, ask questions to help them learn for themselves, and provide useful advice and support without taking over a project.
- » Guide the project team through its first sprint at the beginning of the project and be available to answer questions as needed throughout the project.
- » Work with and relate to the product owner, the development team members, and the scrum master.
- » Be a person from outside a department or organization. Internal agile mentors often come from a company's project management group or center of excellence. If the agile mentor comes from inside the organization, he or she should be able to put aside political considerations when making suggestions and providing advice.

A number of organizations offer agile strategy, planning, and mentorship, including our company, Platinum Edge.

## ***Creating an Environment That Enables Agility***

When you're laying the foundation for adjusting your approach from traditional methods to agile methods, create an environment where agile projects can be successful and project teams can thrive. An agile environment refers to not only physical environments, such as the one we describe in [Chapter 5](#), but also a good organizational environment. To create a good agile project environment, you should have the following:

- » **Good use of agile processes:** This may seem obvious, but using proven agile frameworks and techniques from the beginning. Use the Roadmap to Value in [Figure 16-2](#), using scrum and the other key agile practices to increase your chances of success. Start with the basics; build on them only when the project and your knowledge progress. Progress for the sake of progress doesn't lead to perfection. Remember, practice doesn't make perfect; practice makes permanent. Start out correctly.
- » **Unfettered transparency:** Be open about project status and upcoming process changes. People on the project team and throughout the organization should be privy to project details.
- » **Frequent inspection:** Use the regular feedback loop opportunities that scrum provides to see firsthand how the project is going.
- » **Immediate adaptation:** Follow up on inspection by making necessary changes for improvement throughout the project. Take opportunities to improve today; don't wait until the end of a release or the entire project.
- » **A dedicated scrum team:** Ideally, the product owner, development team, and scrum master will be fully allocated to the project.
- » **A collocated scrum team:** For best results, the product owner, development team, and scrum master should sit together, in the same area of the same office.
- » **A well-trained project team:** When the members of the project team work together to learn about agile values and principles and experiment with agile techniques, they have shared understanding and common expectations about where they're headed as an agile organization.



**FIGURE 16-2:** The Roadmap to Value.

Luckily, many opportunities for training in agile processes are available. You can find formal certification programs as well as non-certification agile courses and workshops. Available agile certifications include the following:

» From the Scrum Alliance:

- Certified ScrumMaster (CSM)
- Advanced Certified ScrumMaster (A-CSM)
- Certified Scrum Product Owner (CSPO)
- Advanced Certified Scrum Product Owner (A-CSPO)
- Certified Scrum Developer (CSD)
- Advanced Certified Scrum Developer (A-CSD)
- Certified Scrum Professional (CSP) for ScrumMasters (CSP-SM), Product Owners (CSP-PO), and Developers (CSP-D)
- Certified Team Coach (CTC)

- Certified Enterprise Coach (CEC)
  - Certified Agile Leadership (CAL)
- » The Project Management Institute Agile Certified Practitioner (PMI-ACP) accreditation
- » From Scrum.org:
- Professional Scrum Master (PSM I, II, III)
  - Professional Scrum Product Owner (PSPO I, II)
  - Professional Scrum Developer (PSD)
- » From the International Consortium for Agile (ICAgile):
- Various tracks in agile coaching, engineering, training, business agility, delivery management, DevOps, enterprise, agility, and value management
- » Numerous university certificate programs

With a good environment, you have a good chance at success.

## ***Support Agility Initially and Over Time***

When you first launch into agile processes, give your agile transition every chance for success by paying attention to key success factors:

- » **Choose a good pilot.** Select a project that's important enough to get everyone's support. At the same time, set expectations: Although the project will produce measurable improvements, the results will be modest while the project team is learning new methods and will improve over time.
- » **Get an agile mentor.** Use a mentor or coach to increase your chances of setting up a good agile environment and maximizing your chances of great performance.
- » **Communicate — a lot.** Keep talking about agile processes at every level of the organization. Use your agile champion to encourage progress through the pilot and toward more extensive agile adaptation.
- » **Prepare to move forward.** Keep thinking ahead. Consider how you'll

take the lessons from the pilot to new projects and teams. Also think about how you'll scale from a single project to many projects, including those with multiple teams.

# Chapter 17

## Scaling across Agile Teams

---

### IN THIS CHAPTER

- » Identifying when and why to scale across multi-team projects
- » Understanding the basics of scaling
- » Exploring scaling challenges

Depending on the schedule, scope, and required skills, many small and medium-sized projects can be accomplished with a single scrum team. Larger projects, however, may require more than one scrum team to achieve the product vision and release goals in a reasonable go-to-market time frame. When more than one scrum team is required, the teams need effective inter-team collaboration, communication, and synchronization — they need to be agile at scale. Regardless of project size, if interdependencies exist between multiple teams working together on the same project, or even across a collection of projects, you may need to scale.



**TIP** Scale only if you have to. Even though you may have the talent and resources available to deploy multiple teams on your project, multiple teams don't automatically ensure higher quality and faster time to market. Always look for ways to implement the tenth agile principle, "Simplicity — the art of maximizing the amount of work not done — is essential." Less is more.

As an agile framework, scrum helps teams organize their work and expose progress effectively, whether your project comprises one scrum team or one thousand scrum teams. Scaling brings new challenges, however, so you want to implement techniques for coordination and collaboration across teams that not only support agile values and principles, but also address the specific challenges facing your project and organization.

In this chapter, we discuss some of the issues to address when you need multiple teams on an agile project. We also provide overviews of some common agile scaling frameworks and approaches that address the challenges of scaling.

## ***Multi-Team Agile Projects***

Organizations determine the need for multiple scrum teams when the product backlog and release plan require a faster speed of development than a single scrum team can achieve.

With agile projects, cross-functional teams work together during every sprint of the project, doing the same types of work each sprint and implementing requirements from the product backlog into completed, working, shippable functionality. When multiple teams work from the same product backlog, however, you have new challenges to address.

Common challenges with more than one scrum team working on the same project include the following:

- » **Project planning:** Agile planning is collaborative, from the beginning. Collaboration for large groups is different than for single scrum teams. Establishing a vision with the broader project team (all scrum teams and stakeholders) and building a product roadmap and product backlog with collaborative input from all parties involved requires a different approach than with single-team projects.
- » **Release planning:** Similar to the challenge of project planning, releases involve more specific planning of scope and release timing. Coordinating who will work on what and when throughout the release cycle is even more critical to ensure that dependencies, scope gaps, and talent allocation match the needs across the project.
- » **Decomposition:** To break down larger requirements in the same backlog, multiple teams may need to be involved in research and refinement discussions and activities. Who initiates these discussions? Who facilitates?
- » **Sprint planning:** Although not the last opportunity to coordinate planning and execution between scrum teams, sprint planning is when

scrum teams lock in a certain amount of scope from the product backlog to execute. At this stage, dependencies between scrum teams become reality. If the preceding activities of developing the product roadmap and release plan have not exposed dependencies, what are some ways scrum teams can expose and address them at sprint planning?

- » **Daily coordination:** Even after effective planning and collaboration from project initiation through sprint planning, scrum teams can and should collaborate each day. Who participates and what can be done while teams are in execution mode?
- » **Sprint review:** With so many teams demonstrating their product increments and seeking feedback, how can stakeholders participate with their limited schedules? How can product owners update the product backlog with all that was learned across multiple scrum teams? How do development teams know what was accomplished by other development teams?
- » **Sprint retrospective:** Multiple scrum teams working together make up a broader project team. How do they identify opportunities for improvement and implement those improvements across the program?
- » **Integration:** All product increments need to work together in an integrated environment. Who does the integration? Who provides the infrastructure to the teams? Who ensures the integrations work?
- » **Architecture decisions:** Who oversees the architecture and technical standards? How can these decisions be decentralized to enable teams to be self-organizing and work as autonomously as possible?

These are some examples; you might be able to identify others based on your experience. Whatever your situation, select solutions to your scaling challenges that address your specific challenge.



TIP Some scaling frameworks offer solutions to challenges you may not have. Be careful not to bloat your framework fixing things that are not broken.



**REMEMBER** Throughout this chapter, we refer to products, projects, programs, and portfolios. In general, a *product* is a set of features that provide some sort of value or usefulness to a customer. A *project* is a planned set of work that requires time, effort, and planning to complete. It has a distinct start and end. A *program* is a collection of projects with an affinity to each other (addressing a certain market segment or rolling into the same product). A *portfolio* is a collection of programs and projects used to meet specific strategic business objectives. The projects or programs of the portfolio may not be directly related to each other but are grouped to facilitate management of the work.

Since the first scrum teams in the mid-1990s, there have been agile projects requiring multiple scrum teams collaborating effectively together. Following are overviews of various scaling frameworks and techniques addressing many of these challenges.

## *Making Work Digestible through Vertical Slicing*

One of the simplest scaling approaches is known as vertical slicing, which provides a straightforward solution for dividing the work across teams so they can incrementally deliver and integrate functionality at every sprint. If your scaling challenge is breaking down the work across teams, vertical slicing is the solution.

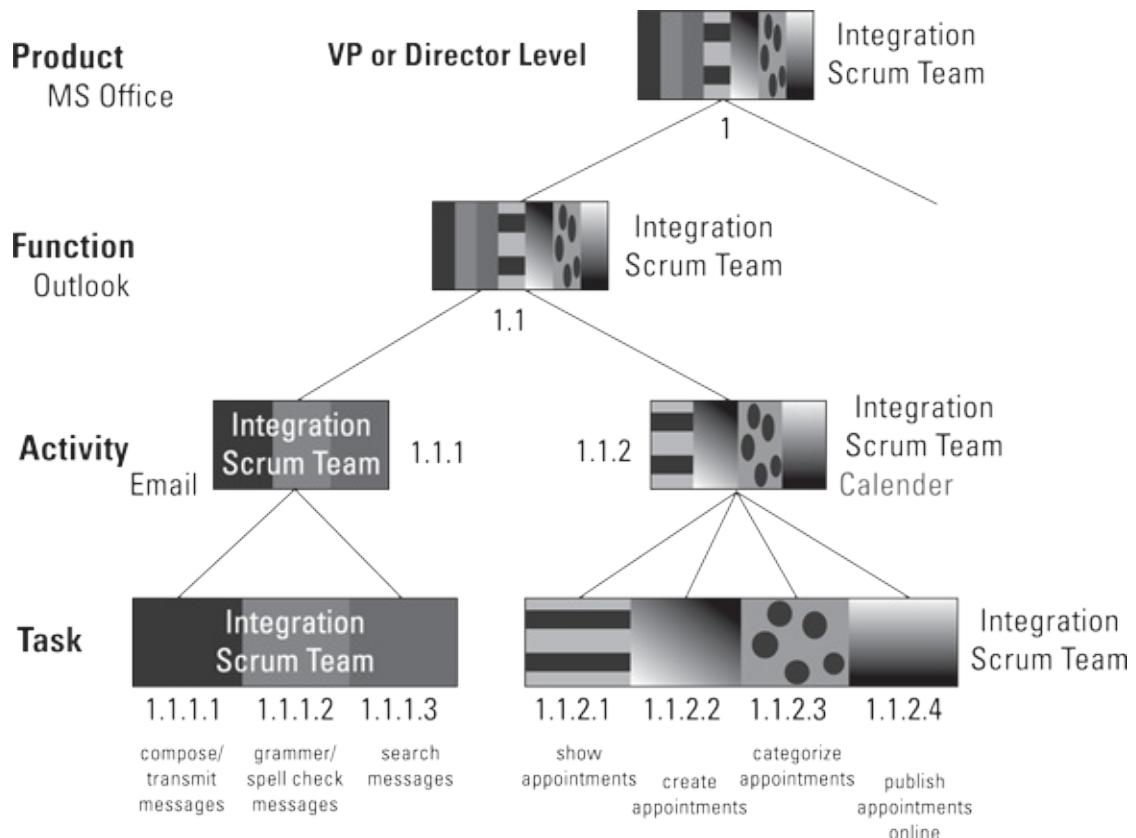


**REMEMBER** The concept of vertical slicing applies to single team projects, too. Development teams consist of people who collectively possess all skills required to turn a requirement into completed, shippable functionality. The development team swarms on one requirement at a time, which is a vertical slice of the product backlog, potentially touching all aspects of the technology and skills required.

With *vertical slicing*, multiple scrum teams work in synchronized sprints of

the same length on a *vertical slice* — a separate portion or module of the overall product — and then those modules are integrated by an integration scrum team after each sprint. The integration scrum team lags the development scrum teams by one sprint and is its own scrum team, with a dedicated product owner, development team members, and scrum master.

[Figure 17-1](#) illustrates how a product backlog is sliced into specific requirements for each scrum team. Then, at the end of each sprint, the individual teams implement working functionality that can be integrated with other working functionality in the broader set of product features. Each individual team's features feed into an integration team's backlog (the scrum team directly above it in the illustration) for architectural and system-level coordination.



**FIGURE 17-1:** Scrum teams working on vertical slices of product features, using Microsoft Office as an example.

The number of integration scrum team levels required depends on the complexity of each project. The figure shows you four levels that a suite of features in Microsoft Office might logically require. (We use Microsoft as an



example because it is familiar to most people.) REMEMBER Each *integration scrum team* handles all system-level development work for the integration of functionality produced by the teams that feed into it, and provides architectural oversight to unify the individual scrum teams.

With Microsoft Office as an illustration:

- » A single scrum team develops the functionality for the Email feature “compose/transmit messages” (requirement ID 1.1.1.1).
- » A different scrum team develops the functionality for “grammar/spell check messages” (1.1.1.2).
- » A third scrum team develops the functionality for “search messages” (1.1.1.3).
- » The integration scrum team (at the Activity level) does the development work to integrate the functionality from these three scrum teams’ functionality into a package (1.1.1) that the Email integration team can integrate into the entire Email module.
- » The Outlook integration team then takes the Email modules, along with the Calendar, Contacts, and other modules, to integrate them into an Outlook package (1.1) that can then be integrated by the MS Office integration team into the entire MS Office suite (1).

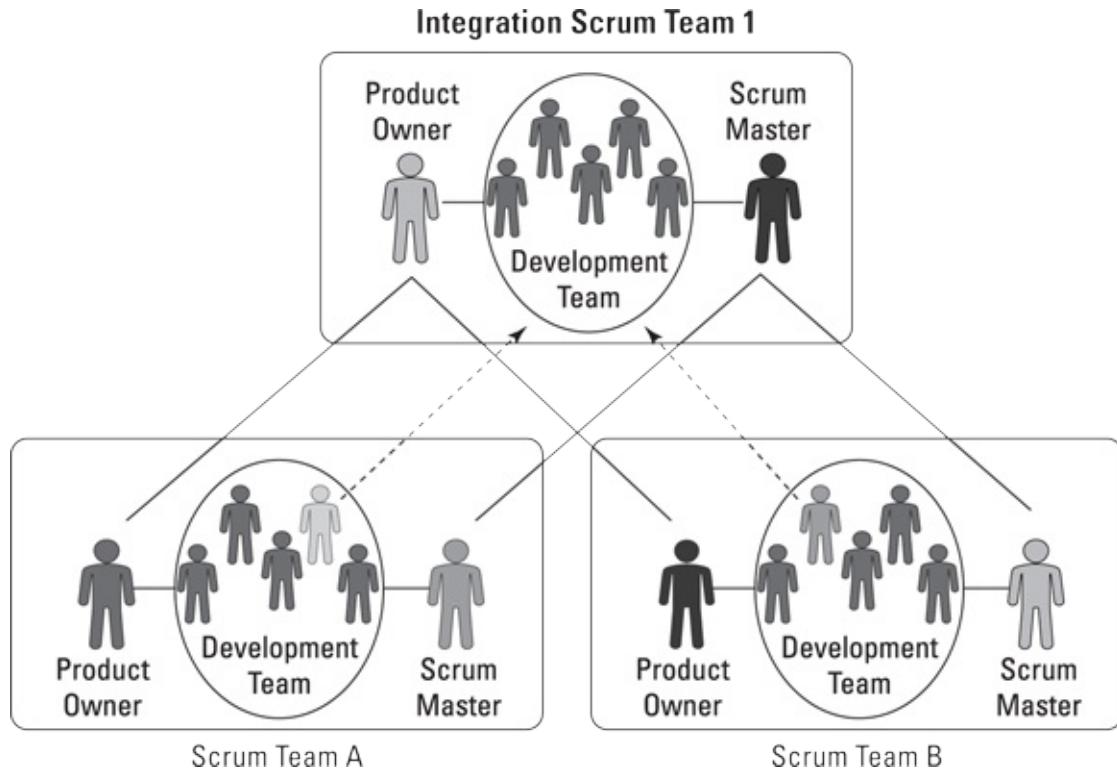
In this example, integration teams operate as separate scrum teams, with dedicated team members for each role.

## ***Scrum of scrums***

How do these different scrum teams coordinate with each other daily? The *scrum of scrums* model facilitates effective integration, coordination, and collaboration among scrum teams using vertical slicing. Almost all scaling frameworks we show you in this chapter use scrum of scrums to enable daily coordination between scrum teams.

[Figure 17-2](#) illustrates each role of each team coordinating daily with people of the same role in other teams regarding priorities, dependencies, and impediments that affect the broader program team. The scrum of scrums for

each role is facilitated by the integration-level person for each role. Thorough integration and release efforts establish a consistent and regular scrum of scrums model.



**FIGURE 17-2:** Scrum of scrums for coordinating between scrum teams.

Each day, individual scrum teams hold their own daily scrums at approximately the same time, in separate locations. Following these daily scrums, the scrum of scrums meetings described next occur.

### ***Product owner scrum of scrums***

Each day, following the individual scrum teams' daily scrums, the product owners from each scrum team meet with the integration team product owner for no longer than 15 minutes. They address the requirements being completed and make any adjustments based on the realities uncovered during the individual scrum team's daily scrum. Each product owner addresses the following:

- » The business requirements that each has accepted or rejected since the last time they met
- » The requirements that should be accepted by the time they meet again

- » Which requirements are impeded and need help from other teams to resolve (such as “John, we won’t be able to do requirement 123 until you complete requirement xyz from your current sprint backlog.”)

The integration team product owner makes the cross-team prioritization decisions necessary to ensure that the impediments are addressed during the daily scrum of scrums.

### ***Development team scrum of scrums***

Each day following the individual scrum teams’ daily scrums, one development team member representative from each scrum team attends the integration team’s daily scrum (which is the scrum of scrums for developers) and participates with the integration development team members in discussing the following:

- » Their team’s accomplishments since the last time they met
- » Their team’s planned accomplishments between now and the next meeting
- » Technical concerns with which they need help
- » Technical direction decisions that the team has made and what anyone should be aware of to prevent potential issues



**TIP** Consider rotating development team members from the individual scrum teams who attend the scrum of scrums (integration team’s daily scrum), either daily or for each sprint, to ensure that everyone stays tuned in to the integration efforts of the portfolio.

### ***Scrum master scrum of scrums***

The scrum masters from each scrum team also meet with the integration scrum team scrum master for no longer than 15 minutes to address the impediments that each team is dealing with. Each scrum master addresses the following:

- » The individual team-level impediments resolved since the last time they met and how they were resolved, in case other scrum masters run into the

issue

- » New impediments identified since last time they met and any unresolved impediments
- » Which impediments they need help resolving
- » Potential impediments that everyone should be aware of

The integration team scrum master then makes sure that escalated impediments are addressed after the daily scrum of scrums.

With vertical slicing, a single product backlog exists, and team attributes are assigned to those requirements as they are broken down and move to the development scrum team. With this model, you can see the overall program and also quickly filter down to your own team's piece of that overall program.

A common question is “Who is responsible for architecture in a vertically sliced program?” The answer is that it depends on which modules will be affected by the decision.



**REMEMBER** Your organization should have existing architectural standards, coding standards, and style guides. This way, each team doesn't have to reinvent the wheel.

Consider an architectural decision that needs to be made and that will affect only module A. The development team of module A would make that decision. If it were going to affect multiple teams, the development team at the integration level that all affected teams roll into would make that decision. That integration level might be one level up or four levels up.

Using [Figure 17-2](#) as an example, an architectural decision affecting two of the Email module teams (1.1.1.2 and 1.1.1.3) would be made by the Email integration team (1.1.1). A decision affecting the search messages Email module team (1.1.1.3) and the Calendar integration scrum team (1.1.2) would be made by the Outlook integration scrum team (1.1).

Vertical slicing is a simple way to maintain the autonomy of each scrum team to deliver valuable functionality within a wider program context. It is also

effective at helping teams have timely and relevant conversations about constraints and progress.

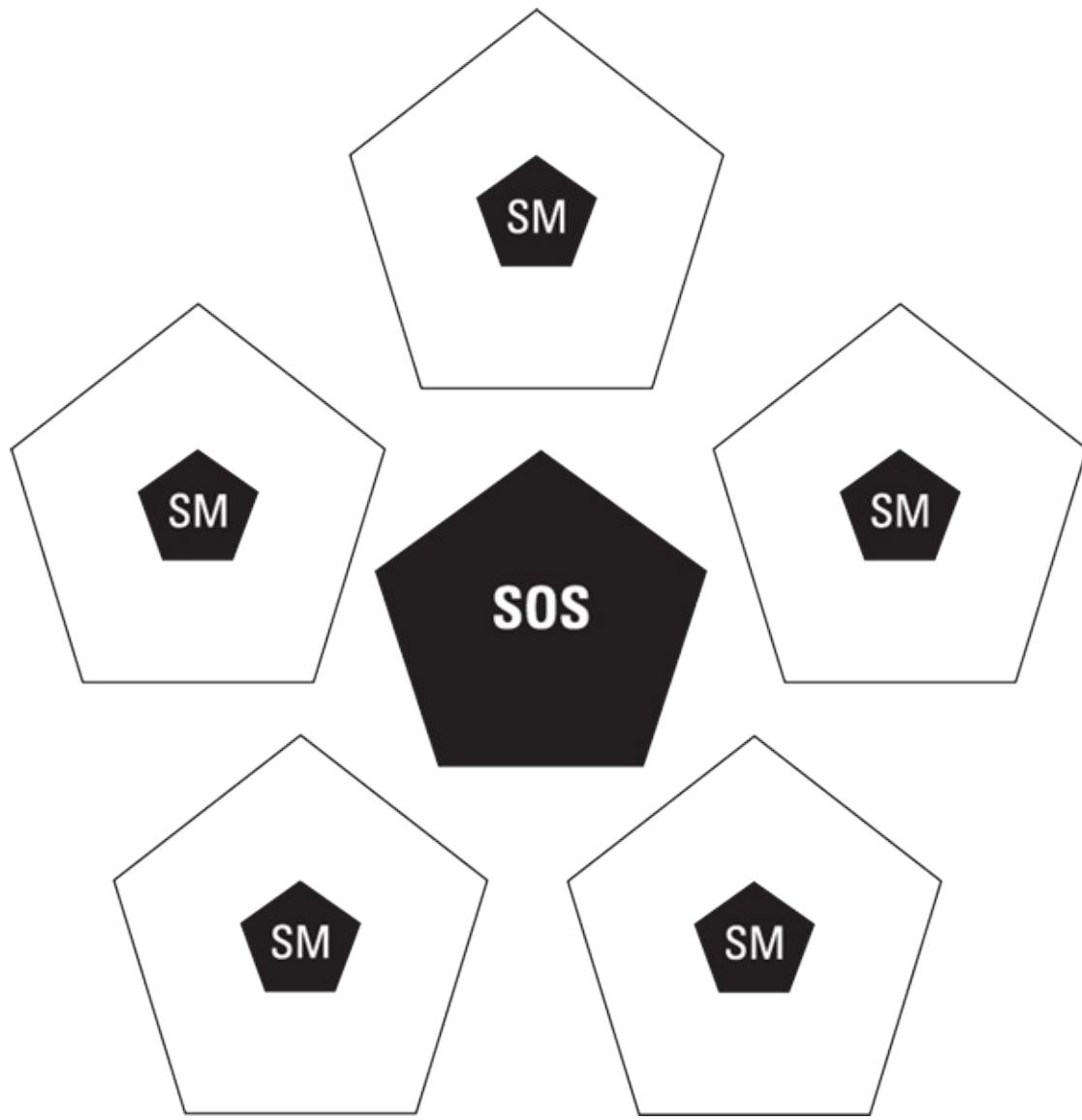
## ***Aligning through Roles with Scrum at Scale***

Agile scaling models vary in complexity and simplicity. The Scrum at Scale approach for two or hundreds of scrum teams working together is a form of basic scrum of scrums model for scrum masters and product owners coordinating communication, impediment removal, priorities, requirement refinement, and planning. Using a scrum of scrums model for the scrum master and product owner roles is how this daily synchronization occurs between teams across programs of varying sizes.

### ***Scaling the scrum master***

Following the vertical slicing model of scrum of scrums, Scrum at Scale groups five scrum masters into a scrum master scrum of scrums. It mirrors the daily scrum for individual scrum teams to surface and remove impediments. With Scrum at Scale, narrowing the scope of a scrum of scrums to five scrum masters from each of five scrum teams limits the communication complexities for effective cross-team collaboration on what the scrum teams are working on and how their work might be affecting each other. A scrum master scrum of scrums coordinates release activities as the release team.

[Figure 17-3](#) illustrates the Scrum at Scale scrum of scrums model.

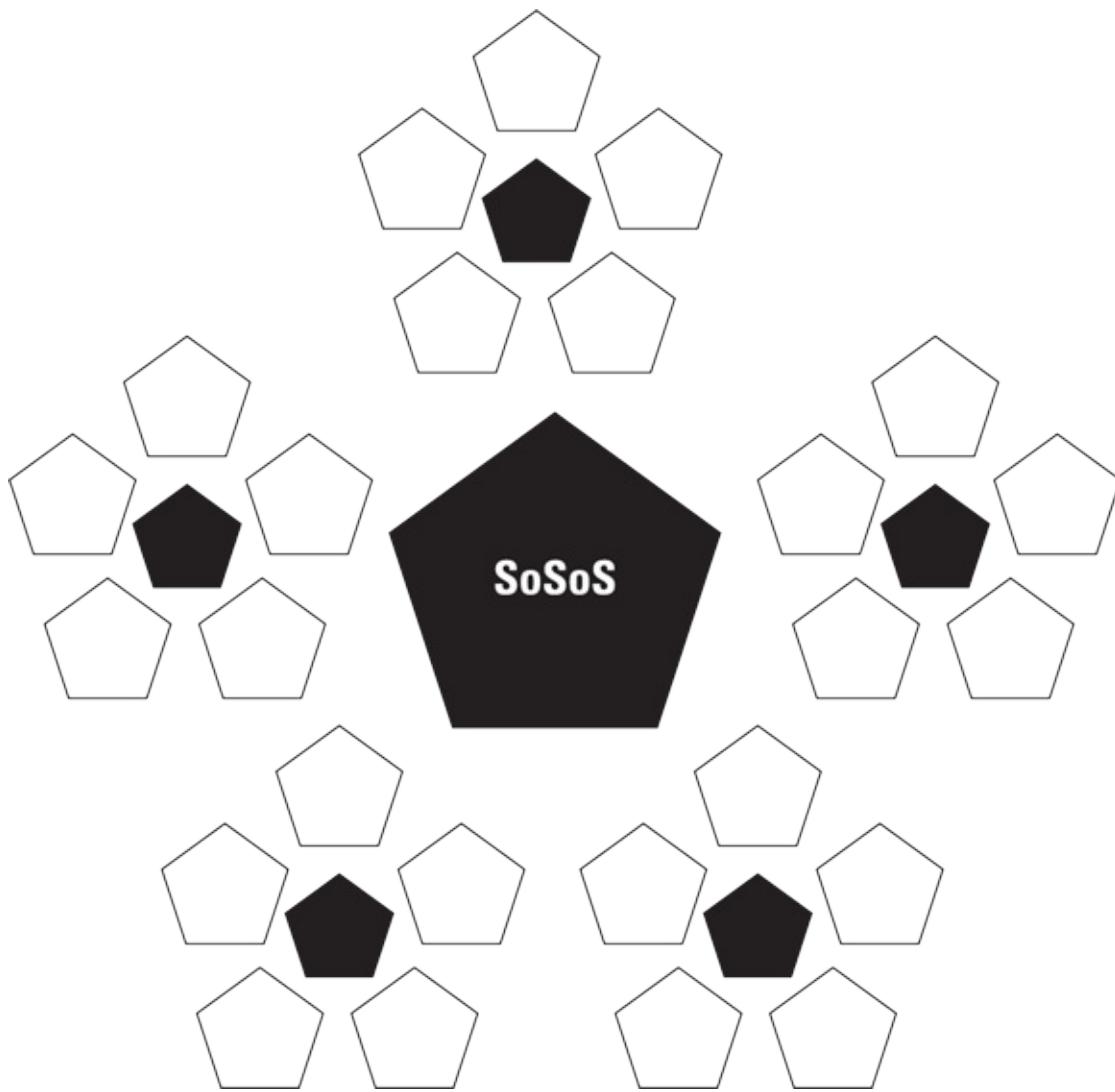


© 1993-2017 Jeff Sutherland & Scrum, Inc.

**FIGURE 17-3:** Scrum at Scale scrum of scrums model.

With projects of more than five scrum teams, Scrum at Scale implements a scrum of scrums of scrums, where a representative of each scrum master scrum of scrums attends with four other scrum of scrums representatives to surface and remove impediments at the scrum of scrums of scrums level.

[Figure 17-4](#) illustrates the Scrum at Scale scrum of scrums of scrums model.

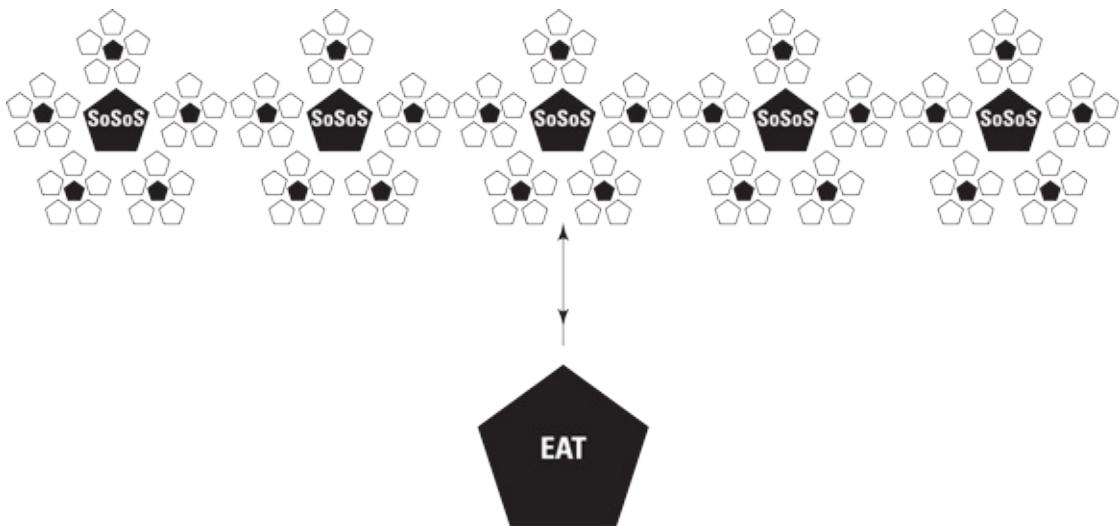


© 1993-2017 Jeff Sutherland & Scrum, Inc.

**FIGURE 17-4:** Scrum at Scale scrum of scrums of scrums model.

When a project has more than 25 teams, an executive action team (EAT) supports a third-level scrum of scrums of scrums to remove the organizational impediments the scrum of scrums groups cannot remove themselves.

[Figure 17-5](#) illustrates the Scrum at Scale third-level scrum of scrums of scrums model with an executive action team (EAT).



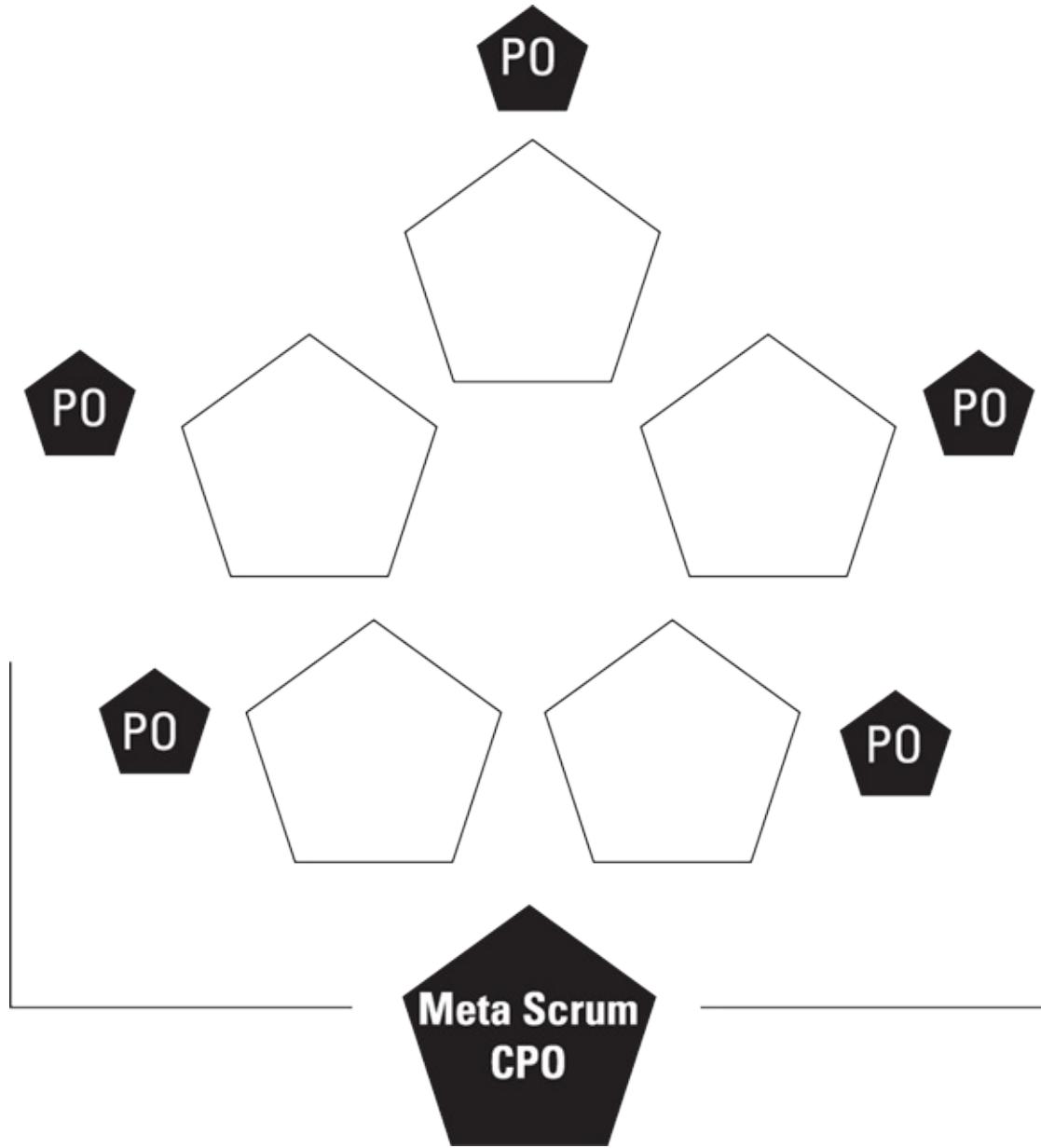
© 1993-2017 Jeff Sutherland & Scrum, Inc.

**FIGURE 17-5:** Scrum at Scale third-level scrum of scrums of scrums model with executive action team (EAT).

## Scaling the product owner

The product owners organize in a similar and aligned way, only instead of labeling them as scrum of scrums, they are meta scrums. A first-level meta scrum brings five product owners together for meta scrum meetings to refine and plan priorities. Each meta scrum has a chief product owner (CPO) who oversees the bigger picture of the vision and product backlog and facilitates the coordination between product owners in the meta scrum.

[Figure 17-6](#) illustrates the Scrum at Scale meta scrum for product owners.

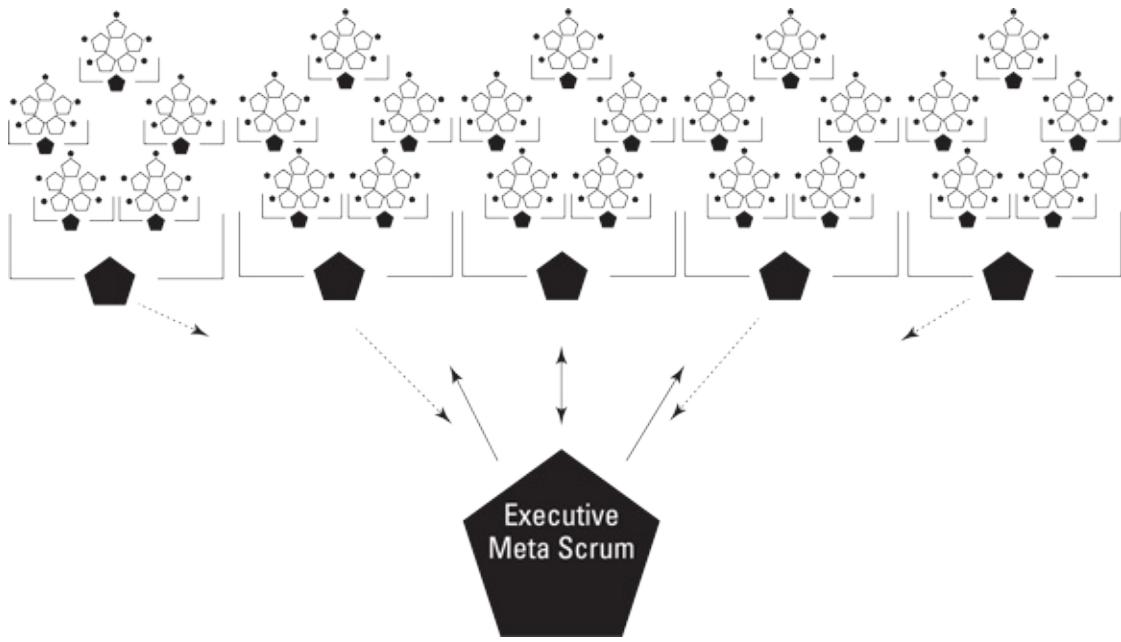


© 1993-2017 Jeff Sutherland & Scrum, Inc.

**FIGURE 17-6:** Scrum at Scale meta scrum for product owners.

At the second-and third-level meta scrums, the grouping aligns with that of the scrum master scrum of scrums of scrums. An executive meta scrum (EMS) supports the meta scrums by owning and communicating the organization-wide vision, taking in technical priority feedback from the meta scrums and providing overall priority decisions for the program.

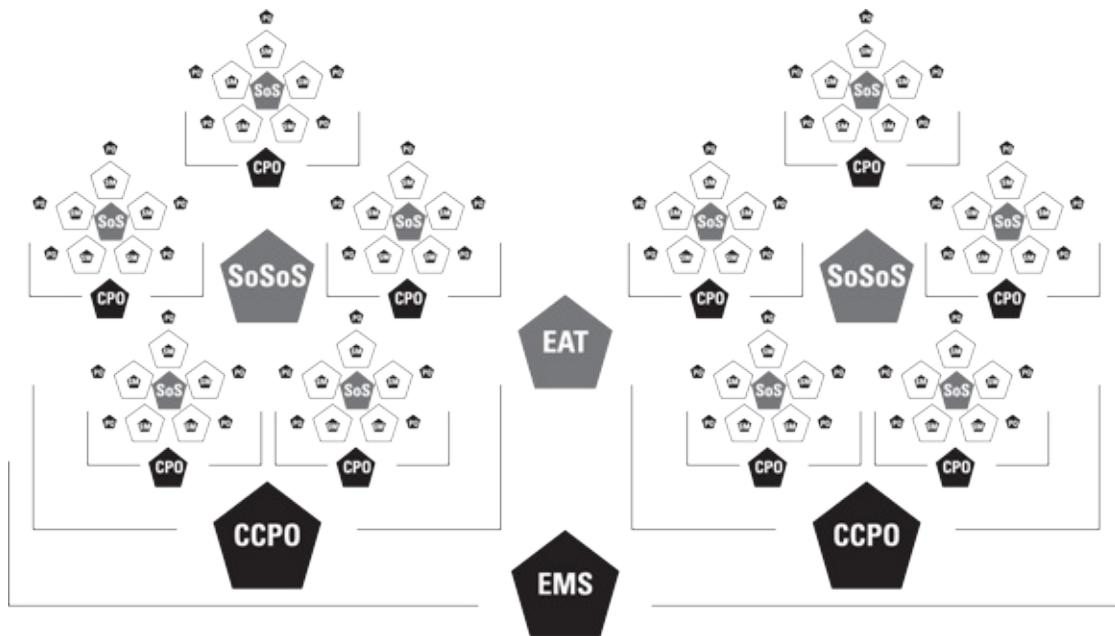
[Figure 17-7](#) illustrates the Scrum at Scale third-level meta scrums model with executive meta scrum (EMS).



© 1993-2017 Jeff Sutherland & Scrum, Inc.

**FIGURE 17-7:** Scrum at Scale third-level meta scrum model with executive meta scrum (EMS).

[Figure 17-8](#) illustrates the aligned grouping of Scrum at Scale's third-level scrum of scrums of scrums and meta scrums model with executive action team (EAT) and executive meta scrum (EMS).



© 1993-2017 Jeff Sutherland & Scrum, Inc.

**FIGURE 17-8:** Alignment of Scrum at Scale third-level scrum of scrums and meta scrums.

A meta scrum should be a synchronization meeting that includes stakeholders. All stakeholders at the CPO level should be present to ensure alignment across the organization and support of the CPO's product backlog prioritization during every sprint.

### ***Synchronizing in one hour a day***

In an hour or less per day, an entire organization can align priorities for the day and accomplish effective coordination of impediment removal. For instance, at 8:00 a.m., each individual scrum team holds their daily scrums separately. At 8:45 a.m., the scrum masters hold their scrum of scrums and the product owners hold their level-one meta scrum meetings. At 9:00 a.m., scrum masters meet in scrum of scrums of scrums, and the product owners meet in level-two meta scrums. Finally, at 9:15 a.m., the scrum master scrum of scrums of scrums meets with the EAT and the product owner meta scrum representatives meets with the EMS.

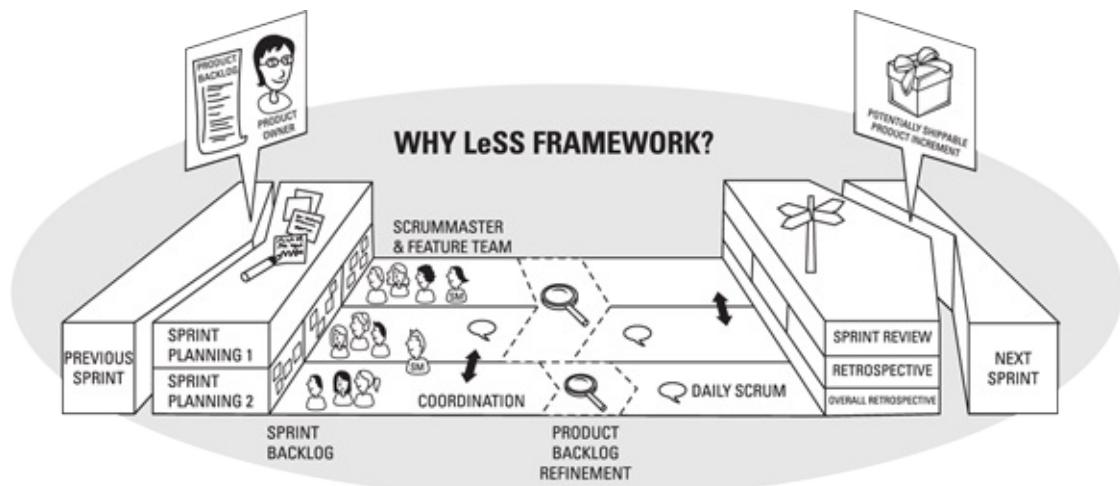
## ***Multi-Team Coordination with LeSS***

Large-scale scrum (LeSS) is another way to scale scrum across massive projects. LeSS is based on principles that support keeping scrum simple when putting multiple scrum teams together to work on the same product backlog. LeSS focuses more on how scrum teams work together than on organizational structures. It also presents a variety of options for addressing each scaling challenge. In this section, we present an overview and then cover a few options that stand out.

LeSS defines two framework sizes: LeSS and LeSS Huge. The difference lies in the size of the total teams involved.

### ***LeSS, the smaller framework***

[Figure 17-9](#) illustrates the basic LeSS framework, using three scrum teams as an example. LeSS recommends no more than eight scrum teams follow the basic model.



*Used with permission, Craig Larman and Bas Vodde.*

**FIGURE 17-9:** Basic LeSS framework.

LeSS outlines how scrum teams work together one sprint at a time, starting with sprint planning, followed by sprint execution and daily scrums, and ending with sprint review and sprint planning. Although much of LeSS remains true to the scrum framework, the following significant differences exist:

- » In LeSS, scrum masters typically work with one to three teams, and there is only one product owner for up to eight teams.



**REMEMBER** We strongly recommend dedicating product owners and scrum masters to each scrum team to ensure that development teams have immediate, direct access to business decisions and clarifications and fast impediment resolution, so they can keep moving without interruption.

- » Sprint planning ([Part 1](#)) does not require all developers to attend, but at least two members per scrum team, along with the product owner, attend. The representative team members then go back and share their information with their teams.
- » Independent sprint planning ([Part 2](#)) and daily scrum meetings occur, and members from different teams can attend each other's meeting to facilitate information sharing.
- » Sprint reviews are usually combined across all teams.

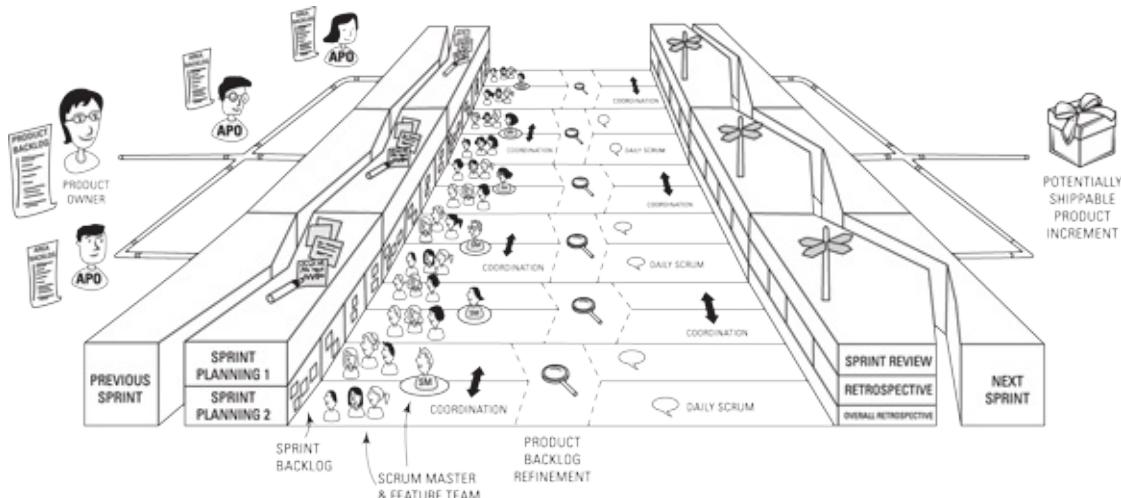
- » Overall sprint retrospectives are held in addition to individual team retrospectives. Scrum masters, product owners, and representatives from development teams inspect and adapt the overall system of the project, such as processes, tools, and communication.

## **LeSS Huge framework**

With LeSS Huge, a few thousand people could work on one project. But the structure remains simple.

The scrum teams are grouped around major areas of customer requirements, called requirement areas. This grouping might be similar to the group of teams who work together under an integration team in vertical slicing.

For each area, you have one area product owner and between four and eight scrum teams (a minimum of four teams in each requirement area prevents too much local optimization and complexity). One overall product owner works with several area product owners, forming a product owner team for the project. [Figure 17-10](#) illustrates LeSS Huge.



Used with permission, Craig Larman and Bas Vodde.

**FIGURE 17-10:** The LeSS Huge framework.

As in scrum at a single team level, as well as in basic LeSS, you have one product backlog, one definition of done, one potentially shippable product increment, one (area) product owner, and one sprint cadence across teams. LeSS Huge is simply a stacking of multiple smaller LeSS implementations for each requirement area.

To enable these teams to work together effectively across the requirement areas

- » The area PO regularly coordinates with each product owner.
- » Requirement areas are added to the product backlog to identify who is planning to work on which parts of the product.
- » A set of parallel sprint meetings is needed per requirement area. Overall sprint reviews and retrospectives involving all teams are necessary to enable continuous inspection and adaptation beyond single teams. These multi-team events help coordinate the overall work and process across the program.

With the exception of limiting opportunities for developers to work closely with business people (the product owner) on a daily basis, LeSS provides a simple way for scaling scrum across projects. We also find the flexibility of coordination techniques suggested in LeSS to be effective for teams addressing their specific multi-team coordination challenges. In addition to a scrum of scrums (discussed earlier in this chapter) and continuous integration (see [Chapter 4](#)), LeSS suggests several options for scrum teams coordinating with other scrum teams, as described in the following sections.

## ***Sprint review bazaar***

Multiple teams work toward the same product increment in each sprint, so all teams have something to demonstrate, and all teams need stakeholder feedback for updating their portion of the product backlog. Because all scrum teams are on the same cadence, even a LeSS basic organization would involve a lot of sprint review meetings for stakeholders to attend on the same day.

LeSS recommends a diverge-converge pattern to the sprint review, similar to a science fair or bazaar format. Each scrum team sets up in one part of a room large enough to accommodate all scrum teams. Each scrum team demonstrates what it did during the sprint, collecting feedback from the stakeholders visiting its area. Stakeholders visit their areas of interest. Scrum teams may loop through their demonstrations a few times to accommodate stakeholders visiting multiple teams. This approach also allows scrum team members to see demonstrations of other scrum teams. Note that combined

sprint reviews can be held in other ways.

Combining sprint reviews increases transparency and collaboration culture across scrum teams.

## ***Observers at the daily scrum***

Although daily scrums are conducted so that the scrum team can coordinate their work for the day, anyone is invited to listen. Transparency is key for agility. The scrum of scrums model described previously in this chapter is a participatory model — developers attending the integration scrum team daily scrum participate in the discussion. However, sometimes other scrum team members just need to be aware of what other teams are doing.

A representative of the development team from one team may attend the daily scrum of another team, observe, and then report back to his or her own team to determine any action to take. This can be a non-disruptive way for other scrum teams to be involved without extra meeting time overhead.

## ***Component communities and mentors***

LeSS takes a vertical slicing approach also to dividing up the product backlog across teams, so multiple teams may "touch" the same system or technology components. For instance, multiple teams may work in a common database, user interface, or automated testing suite. Setting up communities of practice (CoP) around these areas gives these people a chance to collaborate informally on the component areas where they spend most of their time.

CoPs are usually organized by someone from one of the scrum teams who has the knowledge and experience to teach people how the component works, monitor the component long-term, and engage the community in regular discussions, workshops, and reviews of work being done in the component area.

## ***Multi-team meetings***

Similar to the combined sprint review model, LeSS scrum teams may benefit from meeting together for other scrum planning events and activities. Product backlog refinement, sprint planning part two, and other design workshops are some examples. LeSS recommends similar formats for each situation, common elements of which include the following:

- » An overall session first, shared among all teams, to identify which teams are likely to take on which product backlog items.
- » Representatives of each team attend overall sessions (all can attend, but attendance is not required).
- » Team-level sessions follow overall sessions to dive into details.
- » Multi-team breakouts follow overall session, as needed, with just those teams involved.

The key to these sessions is that they are face to face, in the same room, allowing for real-time collaboration to break down dependencies. For distributed LeSS groups (one team in one geographic location and other teams in other locations), videoconferencing is key.

## ***Travelers***

The more versatile your development team, the fewer bottlenecks your scrum team will experience. Traditional organizations have specialists in technical areas, and there are not enough of them to go around to all the scrum teams when starting an agile transition. To begin bridging skill gaps across teams, technical experts can become travelers, joining scrum teams to coach and mentor in their area of expertise through pairing (see [Chapter 4](#)), workshops, and teaching sessions.

As this expertise is shared, the expert mentor continues to lead and grow the skills across the organization (as a CoP organizer). In addition, scrum teams increase their cross-functionality and can develop more efficiently.

## ***Reducing Dependencies with Nexus***

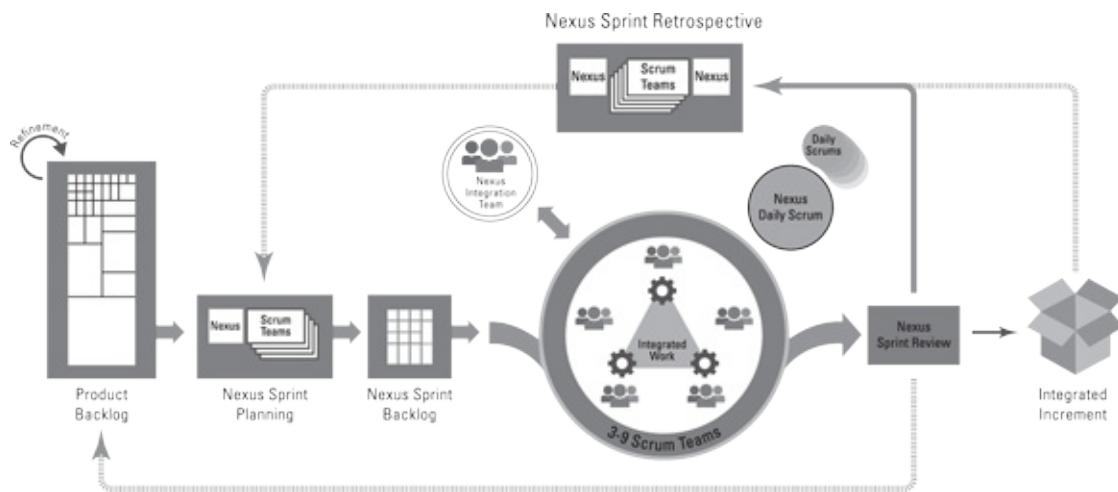
Dependencies between teams working on the same product impede the kind of productivity single scrum teams usually experience without those dependencies. *Nexus* is a scaling framework focused on treating multiple teams as a single unit. Reduction of inter-team dependencies is key to scaling success.

Inter-team dependencies usually revolve around how teams structure requirements and the product backlog, the domain knowledge differences between teams, and the software and test artifacts. Mapping requirements,

team members' knowledge, and test artifacts to the same scrum teams reduces dependencies.

Nexus is a framework that describes how three to nine scrum teams — a *Nexus* — work together on the same product backlog, and under the guidance of a single product owner, to deliver potentially shippable functionality to every sprint.

[Figure 17-11](#) illustrates the Nexus framework.



© 2017 Scrum.org. All rights reserved.

[FIGURE 17-11:](#) The Nexus framework.

In addition to scrum roles, artifacts, and events, Nexus introduces one new role, three new artifacts, and five new events to support the larger group of scrum teams operating together.

Nexus helps scrum teams working on the same product identify and resolve dependencies quickly and early, enabling each scrum team to move forward in their work unblocked and unimpeded. Inter-team dependencies are often created when product backlog items are not sufficiently refined, or not broken down into relatively independent items that can be worked on by a single scrum team. Dependencies can also arise from differences in technical skills or domain knowledge between teams. Joint product backlog refinement helps teams identify dependencies and minimize them before they cause conflict.

## **Nexus role — Nexus integration team**

Similar to the vertical slicing model's integration team concept, the Nexus integration team ensures that an integrated product increment is produced at

least every sprint for the Nexus. The scrum teams do the work, but the Nexus integration team remains accountable for the integrated product as a whole.

The Nexus integration team's activities may include developing tools and practices that will help with integration or serving as coaches and consultants to help with coordination. To accomplish these activities, Nexus integration team members must have a teaching mindset. Their roles are to help expose issues that need to be solved at the Nexus level, and to help the scrum teams solve the issues.

The Nexus integration team consists of people from the member scrum teams of the Nexus. It is a scrum team that consists of the following:

- » **The product owner** is accountable for ordering and refining the Nexus product backlog so that maximum value is derived from the work created by the Nexus each sprint. The product owner's role does not change from scrum; the scope of the work is simply more complex.
- » **Development team members** are usually also members of scrum teams in the Nexus. The priority for the Nexus integration development team members is the Nexus integration team over the individual scrum teams, with the integrated product increment being the prime goal for each sprint. Over time, the members of the Nexus integration team may change depending on specific integration needs over the life of the project.



REMEMBER Dedicating scrum team members to one team eliminates the overhead of frequent cognitive demobilization and remobilization due to context switching. Always be aware of the risks of splitting the focus of team members across multiple teams.

- » **The scrum master** has overall responsibility for ensuring the Nexus framework is enacted and understood. This Nexus integration team scrum master may also be a scrum master in one or more of the other scrum teams in the Nexus.

As a last resort, the Nexus integration team members may pull items from the product backlog to implement them as a scrum team, but they undertake this emergency mode behavior only when all other options have been exhausted

and the scrum teams are not capable of producing an integrated product increment. As the term *emergency mode* suggests, this situation is highly unusual, highly undesirable, and not sustainable. It is undertaken only when it is the only way to help the scrum teams get back on track.

## **Nexus artifacts**

Three additional artifacts provide transparency at the Nexus level for inspection and adaptation:

- » **Nexus goal:** Although the sprint goal is not a separate artifact in scrum, a Nexus sprint goal is explicitly called out. Having a clear, visible, common purpose for all scrum teams in the Nexus is key to keeping all teams in sync throughout the sprint, working toward the integrated product increment.
- » **Nexus sprint backlog:** Each scrum team has its own sprint backlog of implementation and integration tasks. The Nexus sprint backlog is not an aggregation of these sprint backlogs; it exists to expose and map inter-team dependencies and how work is flowing across all scrum teams in the Nexus. The Nexus sprint backlog is updated daily as part of the Nexus daily scrum.
- » **Integrated increment:** All integrated work completed by all the scrum teams in the Nexus during the sprint is the integrated increment. It meets the definition of done for usable, potentially shippable functionality.

## **Nexus events**

Five additional events enhance inter-team coordination of dependencies at the Nexus level.

### **Nexus sprint planning**

During Nexus sprint planning, the product owner provides priority and business context for the sprint, and sets the sprint goal. The individual scrum teams select work for the sprint while highlighting and minimizing dependencies. Each scrum team then holds its own sprint planning to plan the execution of the work it has pulled from the Nexus sprint backlog. Nexus sprint planning concludes when the last scrum team is finished with its individual sprint planning.

## **Nexus daily scrum**

Nexus does not prescribe who should attend the daily scrum — the right people are members of individual scrum teams who understand how their work may affect, or be affected by, the work of other scrum teams. The questions addressed are similar to a single scrum team's daily scrum but focused on cross-team integration, including the following:

- » Did yesterday's work get successfully integrated?
- » What new dependencies have been discovered?
- » What information needs to be shared across teams?

The Nexus daily scrum is held before each scrum team holds its own daily scrum to provide the scrum teams with input to better help them plan their day's work.

## **Nexus sprint review**

Similar to other scaling frameworks, the Nexus sprint review can replace individual scrum team sprint reviews, because the focus is the integrated increment. You can use a variety of techniques for conducting the meeting to maximize stakeholder feedback, but none are prescribed. Techniques suggested in this chapter can be utilized.

## **Nexus sprint retrospective**

The Nexus sprint retrospective is a formal opportunity to improve the way the Nexus works through inspection and adaption. The Nexus sprint retrospective includes three parts:

- » Representatives from Nexus scrum teams meet to identify cross-team issues and make them transparent across the Nexus.
- » Individual scrum teams hold their own sprint retrospectives.
- » Representatives from the scrum teams meet again to decide what to do to resolve Nexus-wide issues.

## **Refinement**

A Nexus uses refinement to decompose product backlog items so they can be developed as independently as possible by a scrum team. In addition to the general process of progressively elaborating requirements we show you in

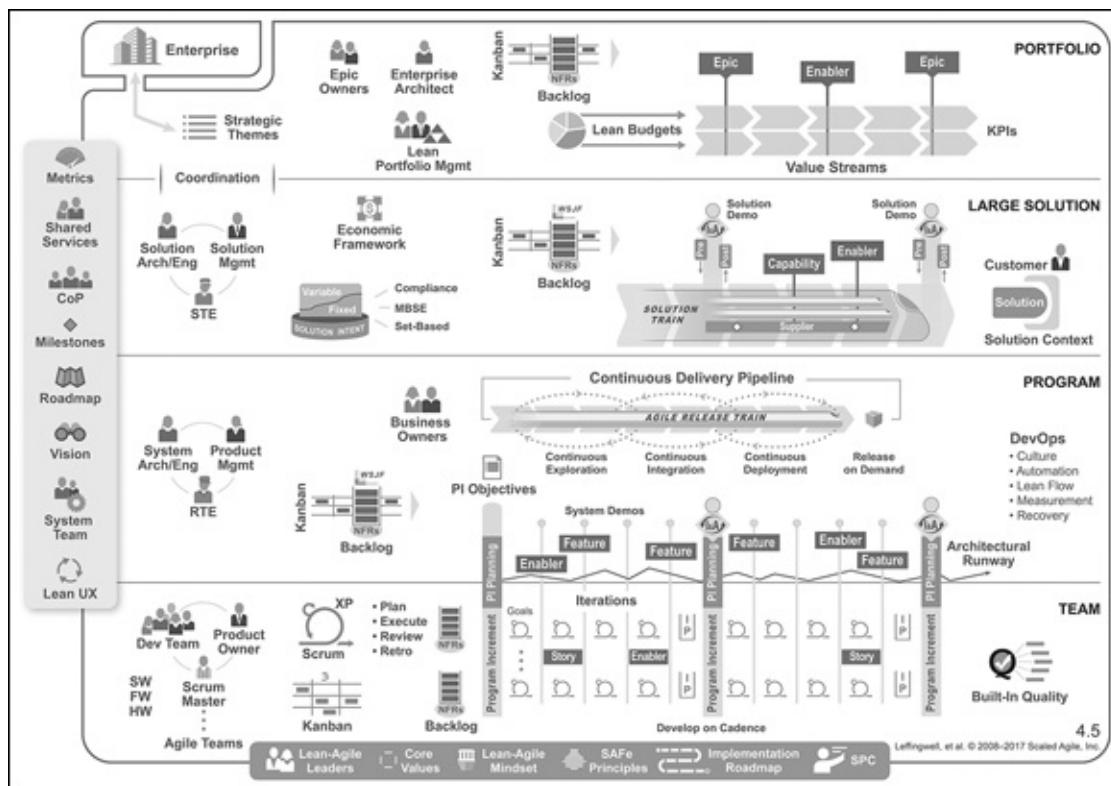
[Chapter 7](#), the Nexus process for refining them includes the following:

- » Breaking product backlog items down enough to understand which scrum teams might be able to implement them
- » Identifying and visualizing dependencies between product backlog items

Nexus is a lightweight framework, focused on extending scrum's empirical approach to products whose development requires more than one scrum team.

## Joint Program Planning with SAFe

*Scaled Agile Framework (SAFe)* is used to scale scrum and agile principles across multiple layers of an IT and software or systems development organization. SAFe addresses scaling at four levels: portfolio, large solutions, program, and team. [Figure 17-12](#) shows the full SAFe 4.5 big picture.

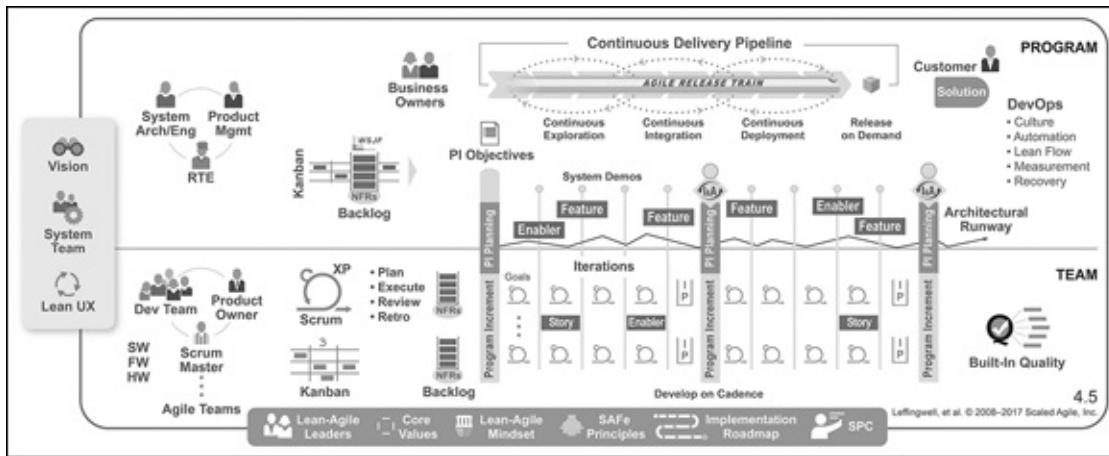


Reproduced with permission from © 2011-2017 Scaled Agile, Inc. All rights reserved. SAFe and Scaled Agile Framework are registered trademarks of Scaled Agile, Inc.

[FIGURE 17-12:](#) SAFe 4.5 for lean software and systems engineering.

SAFe has four configurations, utilizing combinations of the four SAFe levels.

Full SAFe (refer to [Figure 17-12](#)) contains all levels (portfolio, large solution, program, and team). Essential SAFe, shown in [Figure 17-13](#), is a basic starting point for smaller organizations and consists of only the program and team levels. Portfolio SAFe adds the portfolio level to essential SAFe, and is for organizations that have smaller programs within a portfolio. Large solution SAFe adds the large solution level to essential SAFe and is aimed at organizations that are building large solutions that require hundreds of people, but do not require portfolio coordination.



Reproduced with permission from © 2011-2017 Scaled Agile, Inc. All rights reserved. SAFe and Scaled Agile Framework are registered trademarks of Scaled Agile, Inc.

**FIGURE 17-13:** Essential SAFe configuration.

SAFe is underscored by a set of core values, a lean-agile mindset, and the agile values and principles in the Agile Manifesto. Although other scaling frameworks have tactical differences, they also have many similarities, such as the following:

- » Development is done in agile teams.
- » The teams are aligned in sprint length and cadence.
- » A scrum of scrums coordinates at the program level.

We don't go into all details of SAFe here, but we do provide a general overview and highlight a few practices that address some of the scaling challenges discussed previously in this chapter.

## ***Understanding the four SAFe levels***

In SAFe, you find up to four prescribed levels of integration and

coordination, each aimed at decentralizing decisions to the lower levels. We emphasize *up to four* because not all organizations require all levels, such as the large solution level.



**WARNING** Frameworks provide flexibility over rigidity. Although SAFe provides a detailed visualization at all levels of the portfolio organization, avoid implementing structures that are unnecessary for your situation.

The four levels — portfolio, large solution, program, and team — are described next.

## **Portfolio level**

At the *portfolio level*, the vision and roadmap for the entire portfolio are established. Strategic themes are developed to support the vision. Budget, business objectives, and enterprise architecture governance are managed. The portfolio is organized in value streams that align the organization to the value delivered.

SAFe defines a *value stream* as the sequence of steps to deliver value to the customer, from concept to delivery or receipt of payment. It includes the people who do the work, systems, and the flow of materials.

Three portfolio-level roles drive portfolio decisions:

- » **Lean portfolio management (LPM):** This role aligns strategy and execution by communicating strategic themes to the portfolio, establishing value streams, and allocating budgets. LPM is responsible for strategy, investment funding, agile program guidance, and lean governance of the entire portfolio. (Learn more about lean in [Chapter 4](#).) LPM collaborates with many groups across all levels of the organization.
- » **Epic owner:** Epics are introduced in [Chapter 8](#), although SAFe uses a different feature-epic relationship. In SAFe, *epics* are the largest and most long-term initiatives and drive the business value for the organization. They are broken down into features, or capabilities, which are then broken down into user stories that can be executed by single development teams at the team level. Epic owners work with solutions management

and product management at the large solution and program levels, and with agile teams at the team level.

- » **Enterprise architect:** The enterprise architect establishes a common technical vision and drives the holistic approach to technology across programs through continuous feedback, collaboration, and adaptive engineering design and engineering practices.

The portfolio backlog in SAFe consists of both business and enabler epics. Enablers are requirements for extending architectural capabilities to support future business functionality. LPM guides the flow of large initiatives using kanban. (Learn more about kanban in [Chapter 4](#).)

### ***Large solution level***

The *large solution level* hosts the solution train, which is an organizational construct for coordination of multiple agile release trains (ARTs), defined next in the “[Program level](#)” section. The large solution level is for organizations building solutions that require more than 125 people.

Solution trains are coordinated by three roles, similar to the program-level roles, which are all described in the next section, “[Program level](#).”

### ***Program level***

In line with the portfolio vision and backlog, *programs* establish a vision and roadmap to define the outer boundary of their scope of work, focused on selected epics from the portfolio backlog.

At the program level, release and product management occur. This level uses the *agile release train (ART) model*, which is a team of multiple agile teams (50–125 people in total) delivering incremental releases of value. The “train” departs the station on a reliable schedule and features can be loaded onto the train if ready. The ART provides a fixed cadence with which the teams of the program align and synchronize. The rest of the organization, knowing this cadence, can also reliably plan its work around this known release schedule.

If organized at the large solution level, three roles provide coordination for the ARTs in each respective value stream: solution train engineer (STE), solution management, and solution architect/engineer.

Without the large solution level, ARTs are driven by a release train engineer (RTE), product management, and a system architect/engineer. ART roles and

solution train roles are similar, so we explain them together:

- » **Release train engineer (or solution train engineer):** ARTs are generally self-organizing, but they need coordination to steer themselves. RTEs facilitate program level processes, impediment escalation, risk management, and continuous improvement. STEs provide a similar service, working with RTEs guiding the work of all ARTs in the solution train. Similar to scrum masters at the team level, RTEs and VSEs are servant-leaders.
- » **Product management (or solution management):** These people continuously define and prioritize requirements for the ART or solution train so that product owners have the information and empowerment they need to make fast decisions and provide instant clarification to the developers on individual scrum teams.
- » **System architect/engineer (or solution architect/engineer):** A cross-discipline team with system view responsibility for overall architectural and engineering design for the respective ART or solution train. Similar to the way architecture decisions are made at the lowest-level integration team in vertical slicing, the architect/engineer provides the standards to enable developers on the individual scrum teams to make in-the-moment technical decisions.

At this level of integration, the ART works at a cadence of five iterations by default to create what is known as program increments (PI).

### ***Team level***

ARTs are made up of a certain number of individual agile teams, which make up the team level of SAFe. Agile teams in an ART work in cadence with each other, and their team backlogs each support and align with the program vision and backlog.

SAFe has many aspects, but we find the following to be most valuable in addressing challenges of scaling.

### ***Joint program increment planning***

Joint program increment (PI) planning unifies agile teams across an ART. In PI planning, agile teams plan their work for the next PI together, face-to-face, in the same room at the same time.

PI planning includes the following:

- » Setting business context for the PI by a senior executive or business owner.
- » Communicating program vision by product management, and supporting features from the program backlog.
- » Presenting the system architecture vision and any agile-supportive changes to development practices (such as test automation).
- » Outlining the planning process by the RTE.
- » Setting up agile team breakout sessions to determine capacity and backlog items that they will work on in support of the program vision.
- » Reviewing draft plans with all agile teams, with each team presenting key planning outputs, potential risks, and dependencies. Product management and other stakeholders provide input and feedback.
- » Reviewing draft plans by management to identify any issues with scope, talent allocation constraints, and dependencies. Facilitated by the RTE.
- » Breaking out of agile teams to adjust planning based on all feedback.
- » Reviewing the final plan, facilitated by the RTE.

The magic of PI planning is that dependencies are identified and coordinated in the moment during these two-day sessions — two days well spent. If one team identifies a dependency in one of its own requirements during the team breakouts, that team sends a team member to another team to discuss the dependency right there and then. No back-and-forth occurs.

Although no amount of planning can identify every issue upfront, this type of collaboration addresses most issues ahead of time. In addition, it establishes an open line of communication throughout the program increment execution, ensuring teams synchronize and address issues immediately and more effectively than if they had planned as separate teams, sharing documentation without discussion.

## ***Clarity for managers***

In [Chapters 3](#) and [14](#), we discuss ways management changes to enable teams to be more agile and adaptive in nature. For larger organizations, SAFe

provides structure for middle management's involvement with agile teams. The portfolio, large solution, and program levels outline roles and functions not fulfilled by individual team members, providing some clarity to how functional, technical, and other leadership types can clear the way and enable the individual agile teams to be as effective and efficient as possible, as well as connect strategy to execution.

## ***Modular Structures with Enterprise Scrum***

Of all the scaling models presented in this chapter, Enterprise Scrum (ES) may offer the most modular approach to scaling. The ES framework is highly configurable for the sake of achieving overall business agility.

For larger projects, programs, and portfolios, ES stretches the foundations of single-team scrum practice across many teams and supports self-organization at scale through menus of structuring options. These options allow teams of teams to not only track their work of creating functionality but also specify, test, inspect, and adapt everything that matters to their success, including all their configuration choices, at the end of each iteration.

Some configuration menus include patterns for structuring teams and roles, collaboration style, delivery modes, contract types, and a range of metrics. ES also generalizes some of the core elements of scrum (roles, artifacts, and events). We discuss the ES scrum element generalizations and key configuration menus in this section.

### ***ES scrum elements generalizations***

Scaling agility across an organization pulls in people who may not initially be familiar with specific terms used in scrum. ES generalizes some names of scrum roles, artifacts, and events to make them more familiar to members of the broader organization but keeps their functions inline with scrum.



WARNING The elements of scrum — three roles, three artifacts, five events — are central to the scrum framework. Removing or changing any of them

means you're not doing scrum. You don't have to use scrum, but always use the four agile values and 12 agile principles as your litmus test to determine if you're being agile. You can learn more about the agile values and principles in [Chapters 1](#) and [2](#).

Some examples of ES scrum generalizations include the following:

- » A product owner becomes a *business owner* to highlight that this role applies to the overall business, even on initiatives that may not be product focused, but rather service or initiative-focused.
- » The scrum master role becomes a *coach* to emphasize the enabling nature of the role both within the team and externally with all stakeholders and business units.
- » A product backlog becomes a *value list* to emphasize that each value list item may consist of user stories directly affecting functionality, or anything providing value to the business regarding any items outlined in the canvas. Learn about canvases in the next section, "[ES key activities](#)."



TIP The idea of adding more than just product functionality requirements to a product backlog is introduced in [Chapter 7](#), where we show you examples of product backlog items that include not only requirements (functionality) but also maintenance (development on existing functionality), overhead (required work that does not affect functionality), and improvement (action items from sprint retrospectives to improve structures and processes for the team and organization).

- » A sprint backlog becomes a *scrumboard* to emphasize the value of visualizing the work to be done in a sprint on a wall or a task board. You can learn more about task boards in [Chapter 9](#).
- » A sprint becomes a *cycle*. With software projects, cycles are still one to four weeks, with the same cadence of planning, inspecting, and adapting (review and retrospective) as sprints. Non-software cycles may be longer.
- » Sprint planning and sprint review become *cycle planning* and *cycle review*, respectively, to align with the renaming of *sprint*.
- » Sprint retrospective becomes *improvement* to emphasize the forward-

looking direction of inspecting and adapting.

## ES key activities

ES has three key activities, which we outline here by how each applies to scaling across multiple teams.

### Step 1: Visualize everything that matters

ES offers a growing library of templates for organizing project information, including vision, roadmap, roles, teams involved, resources, value list items, deployment methods, stakeholders, and customers. These templates are called *canvases* and are the foundation for teams visualizing their work. [Figure 17-14](#) illustrates the canvas template for a scaled software development project. (ES canvas templates exist for other types of non-software development, but we focus on the scaled software development canvas here.)

High-Level Statement of Vision, Value Proposition, Purpose						
Resources and support needed: * physical * financial * intellectual property * tools * etc.	Business ownership work and role(s)  Coaching work and role(s)	Team(s) work and practices: * team roster(s) * working agreements * definitions of ready & done * technical practices & skills * etc.	Individual scrum team value lists – one per team in vertical slices.  <b>Product increment</b>  <b>Requirements</b>  <b>Architecture</b>  <b>Planning</b>  Each Individual value list (column) can have a different definition of done, as needed.	Delivery channels * marketing * sales * partners * relationships * deployment * etc.	Customers * choosing or discovering * learning about * segmenting * communicating * collaborating * etc.	
Business stakeholders (metrics, reports, relations, etc.)			Customer stakeholders (metrics, reports, relations, etc.)			

Used with permission, © 2017 Enterprise Scrum Inc.

[FIGURE 17-14:](#) ES scaled software development canvas.

Each section contains the work or issues that need to be addressed for each category. The team pulls the items in each section into the value list and subsets of the list onto the scrumboard for implementation during each cycle.

The ES canvas contains everything that matters for successful delivery. It expands the concept of product backlog into a customized value list of every kind of work required. The contents of the value list start at a high level and are refined to whatever level of detail is needed to complete the work in a

cycle. The entire contents of the canvas and value list are reviewed for possible improvements after every cycle.

When scaling with multiple teams, the canvas holds the value list, including all user stories, sliced and detailed as the work progresses (middle section). The value list also includes the full range of surrounding issues and relationships from the canvas, including vision, resources, business ownership, coaching, teams involved, metrics, and interactions with stakeholders, deployment, and customers.

## ***Step 2: Make active configuration choices***

At scale, ES also provides options for how the desired outcomes of some roles, artifacts, or events are achieved to address individual circumstances. Drawing on a set of menus that offer a range of options, teams make active choices about their configurations. Some of the most important menus and choices are described in this section.

### ***DELIVERY TARGETS MENU***

Understanding the level of scale you need is an important first step to determining what approach you need to take. To begin, ES provides a delivery targets menu, which is a set of guidelines for determining the highest level of coordinated delivery that the set of teams in the scaling effort is targeting:

- » Large project: Two or more teams working together to deliver a project
- » Program: Two or more projects serving the same customer segment
- » Portfolio: Two or more application teams working together on multiple programs in a business unit
- » Enterprise architecture: Two or more business areas requiring the use of common architecture elements
- » Business process: Applying agile techniques to non-software organizational units, such as human resources, finance, marketing, sales, compliance, or finance
- » Business agility: Applying agile principles company wide

In contrast to other scaling models that provide a range of multi-team sizes for specific approaches, ES considers two or more teams to be the basis for

considering any options in the menus.

### **STRUCTURAL PATTERNS MENU**

ES focuses on roles rather than titles assigned to individual people. Structural pattern menu options for structuring the business owner and coach roles for each team include the following:

- » Dedicated business owner and dedicated coach for each scrum team (as we outline in [Chapter 6](#))
- » One business owner for all teams and one coach for all teams
- » One business owner for all teams and one coach for each team



- » One business owner for each team and one coach for all teams REMEMBER  
Dedicating each role on a scrum team reduces the risk of lost productivity and defects that often result from task switching, or thrashing.
- » Virtual business owner or virtual coach — a business owner or coach steps into the role as needed, while also filling another role as a developer or an external role to the team
- » Chief business owner or coach to provide direction to individual team business owners or coaches, respectively
- » Virtual teams of business owners or coaches, where either function can be fulfilled by a collaborative, self-managing team of business owners or coaches



**WARNING** Having multiple people in the business owner (product owner) or coach (scrum master) roles increases complexity of communication channels, and may introduce confusion for all project team members.

The configuration you choose depends on many factors, including budget, organizational culture, management style, and individual expertise.

### **COLLABORATION MODES MENU**

The collaboration modes menu is used to coordinate business priorities and

clarification across teams. Various approaches may be used to support or align with the chosen structural pattern:

- » **Centralization:** A business owner makes prioritization decisions and provides clarification to all teams.
- » **Delegation:** A chief business owner provides overall prioritization and meets regularly with individual team business owners to empower them to do the same for their teams for their portion of the value list.
- » **Collaboration:** Each team's business owner works with the other teams' business owners to make collaborative agreements without the oversight of a chief business owner.
- » **Subsumption:** Experts from broader areas of the organization outside developing functionality (such as marketing, finance, and sales) collaborate on everything that the business needs to deliver value to the customer. This scope of collaboration is usually needed to address business process or overall business agility delivery targets outlined previously.

Like other scaling approaches, the scrum of scrums (see the section in this chapter on vertical slicing) is also used in ES to help facilitate the resolution of dependencies across teams, with the encouragement to keep lines of communication always open.

### ***DELIVERY MODES MENU***

Each organization will have different requirements and constraints for how often and on what cadence it delivers functionality to the customer. The frequency and type of delivery will also determine the level of coordination needed between teams on a daily basis, during each cycle, and with each release. Delivery modes menu examples include the following:

- » **Continuous delivery:** The product increment is continuously integrated and tested but not deployed to production.
- » **Continuous deployment:** As each requirement is implemented, it is deployed to production as soon as it is completed.
- » **Cycle:** The product increment is deployed to production at the end of each cycle.

» **Release:** The product increment is deployed after multiple cycles.

Organizations may progress or evolve from one of the delivery modes to another as they inspect and adapt over time, without breaking the ES framework. You can learn more about releasing functionality in [Chapter 8](#).

### ***Step 3: Plan, collaborate, review, and improve everything, every cycle***

ES is about the continuous inspection and adaptation of all aspects of the canvas as well as all configuration choices made from each of the menus. At the end of each cycle, everything completed and everything yet to be done on the canvas is open for inspection and adaptation.

ES also invites teams to consider metrics toward more balanced agile management, and to provide transparency into the team's or program's progress for inspection and adaptation. You can learn more about agile metrics in [Chapter 21](#).

Each of these scaling models has many things in common. They all aim to address the challenges of coordination, communication, prioritization, execution, and integration that come with complex projects and systems requiring more than a single team. Scaling projects across multiple cross-functional feature teams requires coordination and leadership guidance at the highest level of an organization. Management structures need to change from traditional command and control to distributed decision-making and autonomy to the lowest level possible where the work is being executed.

Making this transition requires organization-wide commitment to long-term changes in mindset and structure, which we discuss in [Chapter 18](#).

# Chapter 18

## Being a Change Agent

---

### IN THIS CHAPTER

- » Understanding change management issues and common change management models
- » Following steps for agile adoption in your organization
- » Avoiding common problems in adopting agile
- » Asking the right questions to prevent issues along the way

If you’re contemplating the idea of introducing agile project management to your company or organization, this chapter can help get you started on those changes. Introducing agility means learning and practicing a new mindset, culture, organizational structures, frameworks, and techniques. In this chapter, you learn key principles and steps to implementing agile project management techniques. We also introduce common change models, including the model we use at our company, Platinum Edge. We also cover common pitfalls to avoid in your agile transition.

### *Becoming Agile Requires Change*

Traditional project management is focused on processes, tools, comprehensive documentation, contract negotiation, and following a plan. Although agile project management remains dedicated to addressing each of these, the focus shifts to individuals, interactions, working functionality, customer collaboration, and responding to change.

Waterfall organizations didn’t get where they are overnight and won’t change overnight. For some organizations, decades of forming habits, establishing and protecting fiefdoms, and reinforcing a traditional mindset are engrained. The organizational structure will require some type of change, the leadership will need to learn a new way of looking at developing people and

empowering them to do their work, and those doing the work will have to learn to work together and manage themselves in ways they may not be used to.

## ***Why Change Doesn't Happen on Its Own***

Change is about people more than it is about defining a process. People resist change, and that resistance is based on personal experience, emotion, and fear. We see these reactions firsthand as we help organizations make these changes. Often our first exposure to an organization is when it asks for formal classroom training to learn what it means to be agile and how scrum works. After a two-day class, the level of excitement about implementing this more modern way of thinking and working usually increases, and our students consistently express how much it makes sense.

Scrum is simple. Agile values and principles resonate with almost everyone. But none of it is easy. Scrum for developing products and services is like playing a new game, with new positions, new rules, and a different playing field. Imagine that an American football coach came to his team one day and said, “We’re going to learn how to play futbol (American soccer) today. Meet me out on the pitch in 15 minutes with your gear, and we’ll get right to work.” What would happen? Everyone might know how to play futbol based on what he or she had seen on TV or experienced as a youth, but the team wouldn’t be ready to make the change.

A lot of confusion would ensue. Old rules, techniques, training, and thinking would have to be unlearned for the team to learn the new stuff and come together to compete effectively. Immediately, you’d hear questions from the players, such as the following:

- » When can I use my hands?
- » How many timeout calls do we get?
- » Am I on offense or defense during this play?
- » Where do I line up at kick-off?
- » Who holds the ball when we kick a goal?

- » How many tries do we get before the other team gets the ball?
- » Where's my helmet?
- » These shoes make it hard to kick sometimes.

Transitioning to agile techniques won't happen overnight, but it will happen if you and your organization's leadership take a change management approach to your agile transition. For existing waterfall organizations, agile transformation takes at least one to three years from the time management commits to it. It's an ongoing journey, not a destination.

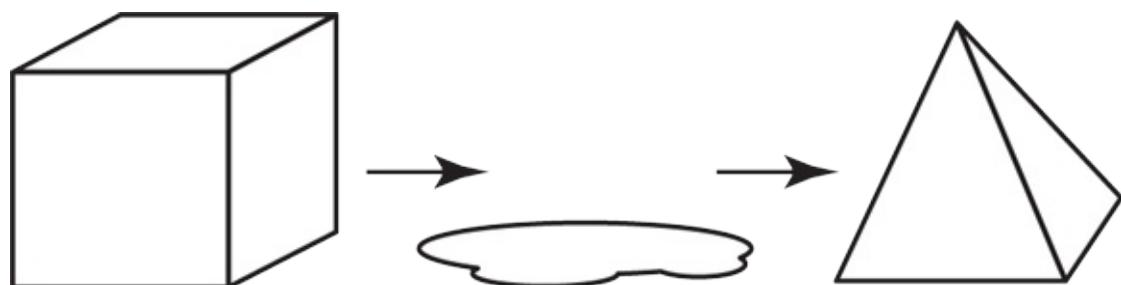
## ***Strategic Approaches to Implementing and Managing Change***

Organizational change initiatives typically fail without a strategy and discipline. Here, we define *failure* as not reaching the desired end state goal of what the organization will look like after the change. Failure is often due to being unclear as to the goal or because the change plan doesn't address the highest risk factors and challenges impeding the desired change.

Various approaches exist to managing change. We show you several here, including ours (Platinum Edge), so you know what to expect as you embark on your own change initiative.

### ***Lewin***

Kurt Lewin was an innovator in social and organizational psychology in the 1940s and established a cornerstone model for understanding effective organizational change. Most modern change models are based on this philosophy, which is unfreeze — change — refreeze, as illustrated in [Figure 18-1](#).



**FIGURE 18-1:** Lewin's unfreeze, change, refreeze change philosophy.

If you want to change the shape of a cube of ice, you first have to change it from its existing frozen state to liquid so that it can be changed or reshaped, then mold the liquid into the new shape you want, then put it through a solidification process to form the new shape. Unfreezing is implied between the first two states in the figure, and the changes made are implied during the unfrozen state.

### ***Unfreeze***

The first stage represents the preparation needed before change can take place — challenging existing beliefs, values, and behaviors. Reexamination and seeking motivation for a new equilibrium is what leads to participation and buy-in for meaningful change.

### ***Change***

The next stage involves uncertainty and resolving that uncertainty to do things a new way. This transitional stage represents the formation of new beliefs, values, and behaviors. Time and communication are the keys to seeing the changes begin to take effect.

### ***Refreeze***

As people embrace new ways, confidence and stability increase, and the change starts to take shape into a solid new process, structure, belief system, or set of behaviors.

This simple pattern provides the foundation for most change management tools and frameworks, including those we discuss in this chapter.

### ***ADKAR's five steps to change***

Prosci is one of the leading organizations in change management and benchmarking research. One of Prosci's change management tools, ADKAR, is an acronym for the five outcomes (awareness, desire, knowledge, ability, and reinforcement) individuals and organizations need to achieve for successful change. It is a goal-oriented model for individuals, and a focus model for the discussions and actions organizations need to take together.

Organizational changes still require change for individuals, so the secret to success is affecting change for everyone involved.

ADKAR outlines the individual's successful journey through change. The five steps of the journey also each align with organizational change activities. They should be completed in the order described next.

### **Awareness**

Humans find change difficult. When change initiatives come top-down in an organization, people may verbally agree to them, but their actions tell a different story. Mismatch of actions and words is usually innocent and natural. Without awareness, or an understanding of the factors influencing management's desire to change, or especially without a recognition that something should change, individuals will not be motivated to change. Informing the individuals in an organization, helping them have a shared understanding of the challenges that exist, and then assessing whether awareness is common constitute the first step to successful, lasting change. It is the basis, without which the initiative won't make progress.

### **Desire**

Based on their awareness of a challenge needing to be addressed, individuals will have an opinion on whether or not change is necessary or desired to address it. Making the connection between the awareness of an issue and what could or should be done about it is the next step. After desire exists for the individuals in an organization, there is motivation to move together to change.

### **Knowledge**

Desire is key, but knowledge of how to make the change and where each individual fits into the change make up the next crucial part of the change process. Individuals throughout the organization need to understand what the changes mean for them, and leadership needs to facilitate education and actions in a cooperative way across the organization. Knowledge comes from training and coaching.

### **Ability**

With new knowledge of how to change, implementation requires acquiring skills, redefining roles, and clearly defining new performance expectations. Other commitments may need to be delayed or replaced with new behaviors or responsibilities. Continued coaching and mentoring may be required, and leadership needs to be clear that this reprioritization of commitments is expected and encouraged.

## **Reinforcement**

Changes don't stick after one successful iteration. New behaviors, skills, and processes must be reinforced through continued corrective action and coaching to ensure that old habits don't return.

The ADKAR model surrounds these steps with assessments and action plans to guide leaders and individuals through their change journey. ADKAR should be used iteratively, using scrum, inspecting and adapting until each step is achieved before progressing to the next step.

## **Kotter's eight steps for leading change**

John Kotter's process for leading change identifies eight common but preventable reasons why organizations fail at their change initiatives, and addresses each with actions that should be taken to successfully lead change.

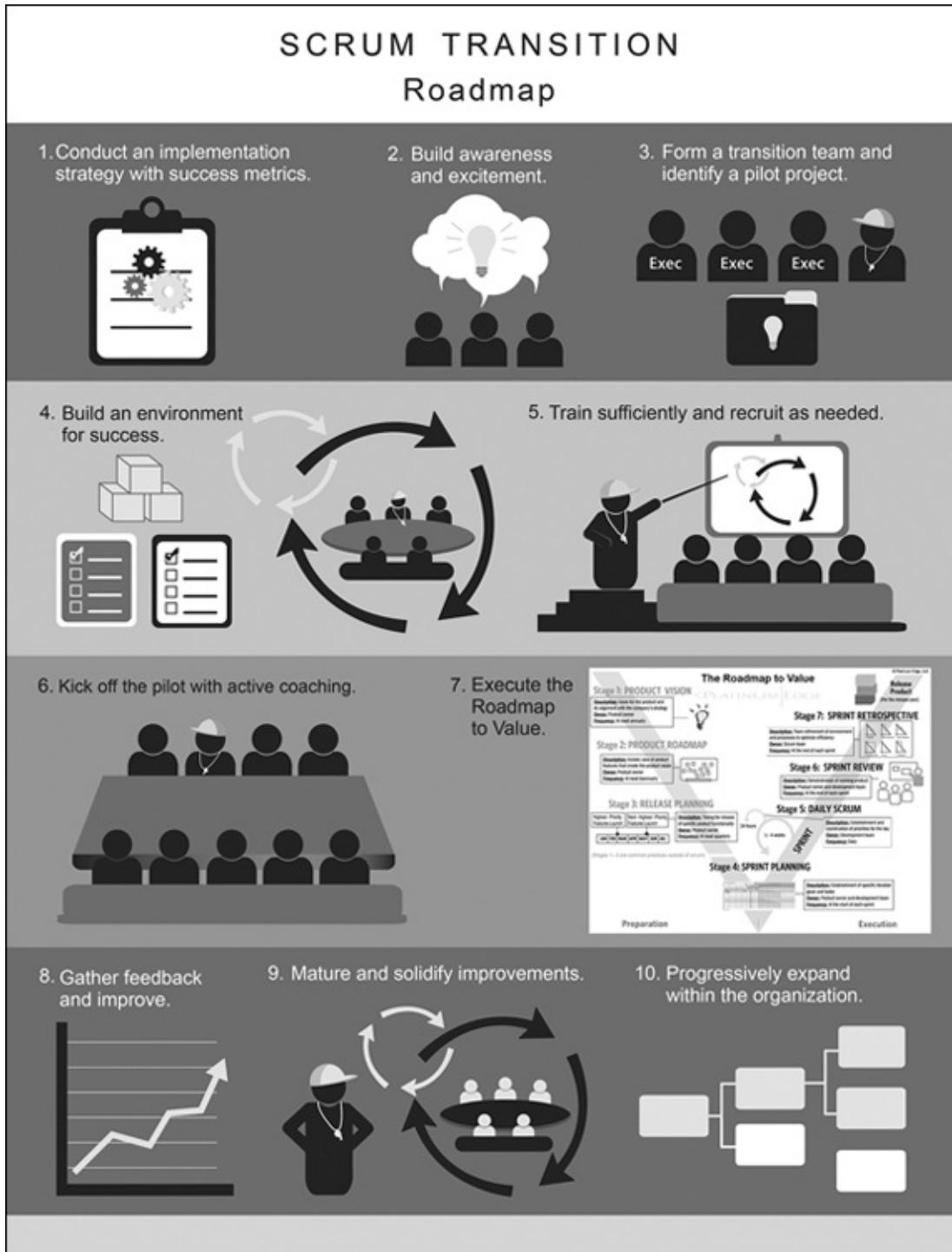
- » **Permitting too much complacency:** The leadership action is to create a sense of urgency. People get used to the status quo, and learn to deal with it. Helping others see the need for change requires the creation of a sense of urgency for change. Leaders must communicate the importance of immediate action.
- » **Lack of a powerful guiding coalition:** The leadership action is to build a guiding coalition. Successful change will require more than just one active supporter, even if that one person is at the highest level of the organization. Executives, directors, managers, and even informal social leaders with influence need to be unified in the need for and vision of a change. This coalition must be formed and drive the change.
- » **Underestimating and undercommunicating the power of vision:** The leadership action is to form a strategic vision and initiatives. Kotter estimates that leadership undercommunicates the vision for change by as much as 1,000 times. Even if people are unhappy with the status quo, they won't always make sacrifices for a change unless they believe in the proposed benefits and that change is possible. As a change coalition, clearly define how the future is different from the past and present, as well as the steps to make that future a reality. We discuss visions and roadmaps for products and services in [Chapter 7](#) — change management also needs to begin with a clear vision of where you're headed.

- » **Lack of rallying around a common opportunity:** The leadership action is to enlist a volunteer army. Change will accelerate and last if people buy in and are internally driven. As a result of leadership's effective communication of vision and need, people should rally around a cause they come to believe in. If they don't rally, reevaluate your messaging, tone, and frequency.
- » **Allowing obstacles to block the vision:** The leadership action is to remove barriers to action. Some obstacles may be only perceived, but others are real. However, both must be overcome. One blocker in the "right" place can be the single reason for failure. Many people tend to avoid confronting obstacles (processes, hierarchies, working across silos), so leadership must act as servant-leaders to identify and remove impediments that are reducing the empowerment of individuals implementing the changes on the front lines.
- » **Lack of short-term wins:** The leadership action is to generate short-term wins. The end transformation goal usually can't be achieved in the short term, so fatigue can set in for everyone involved if successes and progress go unrecognized along the way. Evidence of change should be highlighted and exposed early and regularly. This reinforcement increases morale through difficult times of change, and motivates and encourages continued efforts and progress.
- » **Declaring victory too early:** The leadership action is to sustain acceleration. Celebrating short-term wins sets a false sense of security that change is complete. Each success should build on the previous success. Push on, and push on harder after each success, with increased confidence and credibility. Continue to overcommunicate the vision throughout the transformation.
- » **Neglecting anchoring of changes in organizational culture:** The leadership action is to institute change. Leadership will have the opportunity throughout the change process to connect successes and new behaviors with the culture's evolution and growing strength to keep old habits from returning. These connections should be recognized openly and made visible to everyone as soon as successes and new behaviors are realized.

# ***Platinum Edge's Change Roadmap***

Throughout this book, we highlight the fact that agile processes are different from traditional project management. Moving an organization from waterfall to an agile mindset is a significant change. Through our experience guiding companies through this type of change, we've identified the following important steps to take to successfully become an agile organization.

[Figure 18-2](#) illustrates our agile transition roadmap for successful agile transformation.



**FIGURE 18-2:** Platinum Edge agile transition roadmap.

## ***Step 1: Conduct an implementation strategy with success metrics***

An *implementation strategy* is a plan that outlines the following:

- » Your current strengths to build on as you transition
- » The challenges you'll face based on your current structure
- » Action items for how your organization will transition to agile project management



TIP Implementation strategies are most effectively performed by external agile experts in the form of an assessment or a current state audit.

Whether you engage with a third party or conduct the assessment yourself, make sure the following questions are addressed:

- » **Current processes:** How does your organization run projects today? What does it do well? What are its problems?
- » **Future processes:** How can your company benefit from agile approaches? What agile methods or frameworks will you use? What key changes will your organization need to make? What will your transformed company look like from a team and process perspective?
- » **Step-by-step plan:** How will you move from existing processes to agile processes? What will change immediately? In six months? In a year or longer? This plan should be a roadmap of successive steps getting the company to a sustainable state of agile maturity.
- » **Benefits:** What advantages will the agile transition provide for the people and groups in your organization and the organization as a whole? Agile techniques are a win for most people; identify how they will benefit.
- » **Potential challenges:** What will be the most difficult changes? What departments or people will have the most trouble with agile approaches? Whose fiefdom is being disrupted? What are your potential roadblocks? How will you overcome these challenges?
- » **Success factors:** What organizational factors will help you while switching to agile processes? How will the company commit to a new approach? Which people or departments will be agile champions?

A good implementation strategy will guide your company through its move to

agile practices. A strategy can provide supporters with a clear plan to rally around and support, and it can set realistic expectations for your organization's agile transition.

For your first agile project, identify a quantifiable way to recognize project success. Using metrics will give you a way to instantly demonstrate success to project stakeholders and your organization. Metrics provide specific goals and talking points for sprint retrospectives and help set clear expectations for the project team.



TIP Metrics related to people and performance work best when related to teams rather than to individuals. Scrum teams manage themselves as a team, succeed as a team, fail as a team — and should be evaluated as a team.

Keeping track of project success measurements can do more than help you improve throughout the project. Metrics can provide clear proof of success when you move past your first project and start to scale agile practices throughout your organization.

[Chapter 21](#) describes metrics for success in detail.

## ***Step 2: Build awareness and excitement***

After you have a roadmap showing you the “how” of your agile transition, you need to communicate the coming changes to people in your organization. Agile approaches have many benefits; be sure to let all individuals in your company know about those benefits and get them excited about the coming changes. Here are some ways to build awareness:

- » **Educate people.** People in your organization may not know much — or anything — about agile project management. Educate people about agile principles and approaches and the change that will accompany the new approaches. You can create an agile wiki, hold lunchtime learning sessions, and even have hot-seat discussions (face-to-face discussions with leadership where people can talk safely about concerns and get their questions answered about changes and agile topics) to address concerns with the transition.

- » **Use a variety of communication tools.** Take advantage of communication channels such as newsletters, blogs, intranets, email, and face-to-face workshops to get the word out about the change coming to your organization.
- » **Highlight the benefits.** Make sure people in your company know how an agile approach will help the organization create high-value products, lead to customer satisfaction, and increase employee morale. [Chapter 19](#) has a great list of the benefits of agile project management for this step.
- » **Share the implementation plan.** Make your transition plan available to everyone. Talk about it, both formally and informally. Offer to walk people through it and answer questions. We often print the transition roadmap on posters and distribute it throughout the organization.
- » **Involve the initial scrum team.** As early as you can, let the people who may work on your company's first agile project know about the upcoming changes. Involve the initial scrum team members in planning the transition to help them become enthusiastic agile practitioners.
- » **Be open.** Drive the conversation about new processes. Try to stay ahead of the company rumor mill by speaking openly, answering questions, and quelling myths about agile project management. Structured communications like the hot-seat sessions we mention earlier are a great example of open communication.

Building awareness will generate support for the upcoming changes and alleviate some of the fear that naturally comes with change. Communication will be an important tool to help you successfully implement agile processes.

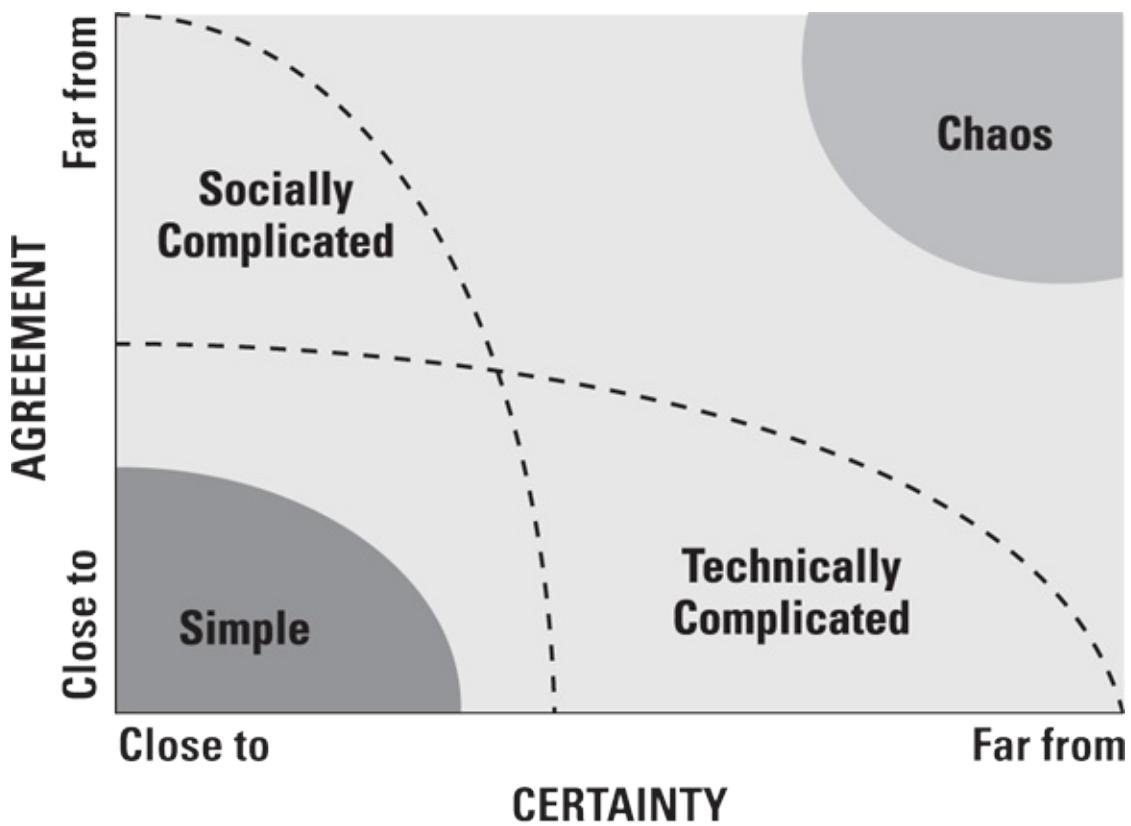
### ***Step 3: Form a transformation team and identify a pilot project***

Identify a team in your company that can be responsible for the agile transformation at the organization level. This agile transition team, which is described in [Chapter 16](#), is made up of executives and other leaders who will systematically improve processes, reporting requirements, and performance measurements across the organization.

The transformation team will create changes within sprints, just like the development team creates product features within sprints. The transformation

team will focus on the highest-priority changes supporting agility in each sprint and will demonstrate its implementation, when possible, during a sprint review with all stakeholders, including the pilot scrum team members.

Starting your agile transition with just one pilot project is a great idea. Having one initial project allows you to figure out how to work with agile methods with little disruption to your organization's overall business. Concentrating on one project to start also lets you work out some of the kinks that inevitably follow change. [Figure 18-3](#) shows the types of projects that benefit most from the agile approach.



Agile benefits are most evident in these conditions

[FIGURE 18-3:](#) Projects that can benefit from agile techniques.

When selecting your first agile project, look for an endeavor with these qualities:

- » **Appropriately important:** Make sure the project you choose is important enough to merit interest within your company. However, avoid the most important project coming up; you want room to make and learn

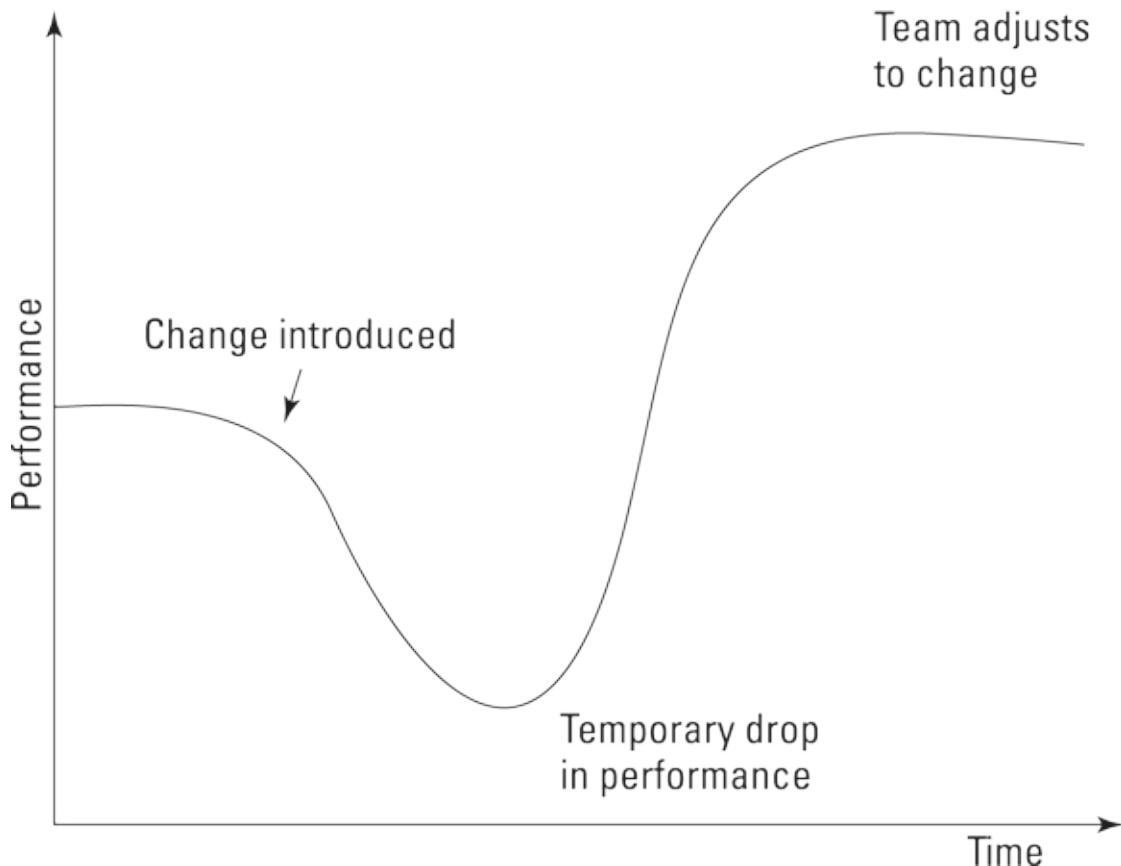
from mistakes. See the note on the blame game in the later section "[Avoiding Pitfalls](#)."

- » **Sufficiently visible:** Your pilot project should be visible to your organization's key influencers, but don't make it the most high-profile item on the agenda. You will need the freedom to adjust to new processes; critical projects may not allow for that freedom on the first try of a new approach.
- » **Clear and containable:** Look for a product with clear requirements and a business group that can commit to defining and prioritizing those requirements. Try to choose a project that has a distinct end point, rather than one that can expand indefinitely.
- » **Not too large:** Select a project that you can complete with no more than two scrum teams working simultaneously to prevent too many moving parts at once.
- » **Tangibly measurable:** Choose a project that you know can show measurable value within sprints.



TECHNICAL STUFF

People need time to adjust to organizational changes of any type, not just agile transitions. Studies have found that with large changes, companies and teams will see dips in performance before they see improvements. *Satir's Curve*, shown in [Figure 18-4](#), illustrates the process of teams' excitement, chaos, and finally adjustment to new processes.



**FIGURE 18-4:** Satir's Curve.

After you've successfully run one agile project, you'll have a foundation for future successes.

### ***Step 4: Build an environment for success***

One of the agile principles states, “Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.”

We outline what it means to create an environment to enable success in [Chapter 5](#). Study the 4 agile values and the 12 agile principles carefully (see [Chapter 2](#)) and seriously to determine whether you’re creating an environment for success or rationalizing that the status quo is good enough.

Start fixing and improving your environment as early as possible.

### ***Step 5: Train sufficiently and recruit as needed***

Training is a critical step when shifting to an agile mindset. The combination of face-to-face training with experienced agile experts and the ability to work

through exercises using agile processes is the best way to help the project team to absorb and develop the knowledge needed to successfully begin an agile project.

Training works best when the members of the project team can train and learn together. As agile trainers and mentors, we've had the opportunity to overhear conversations between project team members that start, "Remember when Mark showed us how to ...? That worked when we did it in class. Let's try it and see what happens." If the product owner, development team, scrum master, and project stakeholders can attend the same class, they can apply lessons to their work as a team.

Recruiting talent to fill gaps in the roles you need avoids the obvious problems you'll have at the start of the transition. Without a dedicated product owner and his or her clear direction to the team, how likely is your project to succeed? How will that affect the team's ability to self-organize? Who will facilitate the many interactions if you're missing a scrum master? What will the first sprint look like if you're missing a key skill on the development team required to minimally achieve the first sprint goal?

Work with your human resources department as early as possible to start the recruiting process. Work with your agile expert advisers to tap into their network of experienced agile practitioners.

## ***Step 6: Kick off the pilot with active coaching***

When you have a clear agile implementation strategy, an excited and trained project team, a pilot project with a product backlog, and clear measures for success, congratulations! You're ready to run your first sprint.

Don't forget, though — agile approaches are new to the pilot team. Teams need coaching to become high performing. Engage with agile experts for agile coaching to start the project right.



TIP Practice doesn't make perfect. Practice makes permanent.

As the scrum team plans its first sprint, it should not bite off too many requirements. Keep in mind that you're just starting to learn about a new process and a new product. New scrum teams often take on a smaller amount

of work than they think they can complete in their first sprints. A typical progression follows.



**REMEMBER** After you establish overall goals through the product's vision statement, product roadmap, and initial release goal, your product backlog needs only enough user-story level requirements (see [Chapter 8](#)) for one sprint for the scrum team to start development.

- » **In sprint 1**, scrum teams take on 25 percent of the work they think they can complete during sprint planning.
- » **In sprint 2**, scrum teams take on 50 percent of the work they think they can complete during sprint planning.
- » **In sprint 3**, scrum teams take on 75 percent of the work they think they can complete during sprint planning.
- » **In sprint 4 and beyond**, scrum teams take on 100 percent of the work they think they can complete during sprint planning.

By sprint 4, the scrum team will be more comfortable with new processes, will know more about the product, and will be able to estimate tasks with more accuracy.



**WARNING** You can't plan away uncertainty. Don't fall victim to analysis paralysis; set a direction and go!

Throughout the first sprint, be sure to consciously stick with agile practices. Think about the following during your first sprint:

- » Have your daily scrum meeting, even if you feel like you didn't make any progress. Remember to state roadblocks, too!
- » The development team may need to remember to manage itself and not look to the product owner, the scrum master, or anywhere besides the sprint backlog for task assignments.
- » The scrum master may have to remember to protect the development team

from outside work and distractions, especially while other members of the organization get used to having a dedicated agile project team around.

- » The product owner may have to become accustomed to working directly with the development team, being available for questions, and reviewing and accepting completed requirements immediately.

In the first sprint, expect the road to be a little bumpy. That's okay; agile processes are about learning and adapting.

In [Chapter 8](#), you can see how the scrum team can plan the sprint. [Chapter 9](#) provides the day-to-day details on running the sprint.

## ***Step 7: Execute the Roadmap to Value***

When you've chosen your pilot project, don't fall into the trap of using a plan from an old methodology or set of habits. Instead, use agile processes from the project's start.

We outline the Roadmap to Value throughout this book, introducing it in [Chapter 7](#) and leading you through each of the seven stages in [Chapters 7](#) through [10](#).

## ***Step 8: Gather feedback and improve***

You'll naturally make mistakes at first. No problem. At the end of your first sprint, you gather feedback and improve with two important events: the sprint review and the sprint retrospective.

In your first sprint review, it will be important for the product owner to set expectations about the format of the meeting, along with the sprint goal and completed product functionality. The sprint review is about product demonstration — fancy presentations and handouts are unnecessary overhead. Project stakeholders may initially be taken aback by a bare-bones approach. However, those stakeholders will soon be impressed as they find a working product replacing the fluff of slides and lists. Transparency and visibility — show, rather than tell.

The first sprint retrospective may require setting some expectations as well. It will help to conduct the meeting with a preset format, such as the one in [Chapter 10](#), both to spark conversation and avoid a free-for-all complaining session.

In your first sprint retrospective, pay extra attention to the following:

- » Keep in mind how well you met the sprint goal, not how many user stories you completed.
- » Go over how well you completed requirements to meet the definition of done: defined, tested, integrated, and documented.
- » Discuss how you met your project success metrics.
- » Talk about how well you stuck with agile principles. We start the journey with principles.
- » Celebrate successes, even small gains, as well as examine problems and solutions.
- » Remember that the scrum team should manage the meeting as a team, gain consensus on how to improve, and leave the meeting with a plan of action.

You can find more details about both sprint reviews and sprint retrospectives in [Chapter 10](#).

## ***Step 9: Mature and solidify improvements***

Inspecting and adapting enables scrum teams to grow as a team and to mature with each sprint.

Agile practitioners sometimes compare the process of maturing with the martial arts learning technique of *Shu Ha Ri*, a Japanese term that can be translated to “maintain, detach, transcend.” The term describes three stages in which people learn new skills:

- » **In the *Shu* stage,** students follow a new skill as they were taught, without deviation, to commit that skill to memory and make it automatic.  
New scrum teams can benefit from making a habit of closely following agile processes, until those processes become familiar. During the Shu stage, scrum teams may work closely with an agile coach or mentor to follow processes correctly.
- » **In the *Ha* stage,** students start to improvise as they understand more about how their new skill works. Sometimes the improvisations will work, and sometimes they won’t. The students will learn more about the

skill from these successes and failures.

As scrum teams understand more about how agile approaches work, they may try variations on processes for their own project. During the Ha stage, scrum teams will find that the sprint retrospective is a valuable tool for talking about how their improvisations worked or did not work. In this stage, scrum team members may still learn from an agile mentor, but they may also learn from one another, from other agile professionals, and from starting to teach agile skills to others.

- » **In the *Ri* stage,** the skill comes naturally to the former student, who will know what works and what doesn't. The former student can now innovate with confidence.

With practice, scrum teams will get to the point where agile processes are easy and comfortable, like riding a bicycle or driving a car. In the Ri stage, scrum teams can customize processes, knowing what works in the spirit of the Agile Manifesto and 12 Agile Principles.

At first, maturing as a scrum team can take a concentrated effort and commitment to using agile processes and upholding agile values. Eventually, however, the scrum team will be humming along, improving from sprint to sprint, and inspiring others throughout the organization.

With time, as scrum teams and project stakeholders mature, entire companies can mature into successful agile organizations.

## ***Step 10: Progressively expand within the organization***

Completing a successful project is an important step in moving an organization to agile project management. With metrics that prove the success of your project and the value of agile methodologies, you can garner commitment from your company to support new agile projects.

To progressively scale agile project management across an organization, start with the following:

- » **Seed new teams.** An agile project team that has reached maturity — the people who worked on the first agile project — should now have the expertise and enthusiasm to become agile ambassadors in the organization. These people can join new agile project teams and help those teams learn and grow.

- » **Redefine metrics.** Identify measurements for success, across the organization, with each new scrum team and with each new project.
- » **Scale methodically.** It can be exciting to produce great results, but companywide improvements require significant process changes. Don't move faster than the organization can handle. Check out [Chapter 17](#) for different ways of scaling agile projects across multiple teams.
- » **Identify new challenges.** Your first agile project may have uncovered roadblocks that you didn't consider in your original implementation plan. Update your strategy and maturity roadmap as needed.
- » **Continue learning.** As you roll out new processes, make sure that new team members have the proper training, mentorship, and resources to effectively run agile projects.

The preceding steps work for successful agile project management transitions. Use these steps and return to them as you scale, and you can make agile practices thrive in your organization.

## Avoiding Pitfalls

Project teams can make a number of common but serious mistakes when implementing agile practices. [Table 18-1](#) provides an overview of some typical problems and ways to turn them around.

**TABLE 18-1** Common Agile Transition Problems and Solutions

Problem	Description	Potential Solution
Faux agile or double work agile or both	<p>Sometimes organizations will say that they are "doing agile." They may go through some of the practices used on agile projects, but they haven't embraced agile principles and continue creating waterfall deliverables and products. This is sometimes called <i>faux agile</i> and is a sure path to avoiding the benefits of agile techniques.</p> <p>Trying to complete agile processes in addition to waterfall processes, documents, and meetings is another faux agile approach. <i>Double work agile</i> results in quick project team burnout. If you're doing twice the work, you aren't adhering</p>	Insist on following one process — an agile process. Garner support from management to avoid non-agile principles and practices.

to agile principles.

---

Lack of training	<p>Investment in a hands-on training class will provide a quicker, better learning environment than even the best book, video, blog, or white paper. Lack of training often indicates an overall lack of organizational commitment to agile practices.</p> <p>Keep in mind that training can help scrum teams avoid many of the mistakes on this list.</p>	<p>Build training into your implementation strategy. Giving teams the right foundation of skills is critical to success and necessary at the start of your agile transition.</p>
Ineffective product owner	<p>The product owner role is non-traditional. Agile project teams need a product owner who is an expert on business needs and priorities and can work well with the rest of the scrum team on a daily basis. An absent or indecisive product owner will quickly sink an agile project.</p>	<p>Start the project with a person who has the time, expertise, and temperament to be a good product owner.</p> <p>Ensure the product owner has proper training. The scrum master can help coach the product owner and may try to clear roadblocks preventing the product owner from being effective. If removing impediments doesn't work, the scrum team should insist on replacing the ineffective product owner with a product owner — or at least an agent — who can make product decisions and help the scrum team be successful.</p>
Lack of automated testing	<p>Without automated testing, it may be impossible to fully complete and test work within a sprint. Manual testing requires time that fast-moving scrum teams don't have.</p>	<p>You can find many low-cost, open-source testing tools on the market today. Look into the right tools and make a commitment as a development team to using those tools.</p>
Lack of transition support	<p>Making the transition successfully is difficult and far from guaranteed. It pays to do it right the first time with people who know what they are doing.</p>	<p>When you decide to move to agile project management, enlist the help of an agile mentor — either internally from your organization or externally from a consulting firm — who can support your transition.</p> <p>Process is easy, but people are hard. It pays to invest in professional transition support with an experienced partner who understands behavioral science and organizational change.</p>
Inappropriate physical environment	<p>When scrum teams are not collocated, they lose the advantage of face-to-face communication. Being in the same building isn't enough; scrum teams need to sit together in the same area.</p>	<p>If your scrum team is in the same building but not sitting in the same area, move the team together.</p> <p>Consider creating a room or annex for the scrum team to continually collaborate.</p> <p>Try to keep the scrum team area away from distractors, such as the guy who can talk forever or the manager who needs just one small favor.</p> <p>Before starting a project with a dislocated scrum team, do what you can to enlist local talent. If you must work with a dislocated scrum team, take a look at <a href="#">Chapter 14</a> to see how to manage dislocated teams.</p>
	<p>Scrum team members who don't support agile processes, don't work well with</p>	<p>When creating a scrum team, consider how well potential team members will accept the agile</p>

Poor team selection	others, or don't have capacity for self-management will sabotage a new agile project from within.	potential team members will enact the agile principles. The keys are versatility and a willingness to learn.
Discipline slips	Remember that agile projects still need requirements, design, development, testing, and releases. Doing that work in sprints requires discipline.	You need more, not less, discipline to deliver working functionality in a short iteration. Progress needs to be consistent and constant. The daily scrum helps ensure that progress is occurring throughout the sprint. Use the sprint retrospective as an opportunity to reset approaches to discipline.
Lack of support for learning	Scrum teams succeed as teams and fail as teams; calling out one person's mistakes (known as the <i>blame game</i> ) destroys the learning environment and destroys innovation.	The scrum team can make a commitment at the project start to leaving room for learning and to accepting success and failures as a group.
Diluting until dead	Watering down agile processes with old waterfall habits erodes the benefits of agile processes until those benefits no longer exist.	When making process changes, stop and consider whether those changes support the Agile Manifesto and the 12 Agile Principles. Resist changes that don't work with the manifesto and principles. Remember to maximize work not done.

As you may notice, many of these pitfalls are related to a lack of organizational support, the need for training, and falling back on old project management practices. If your company supports positive changes, if the project team is trained, and if the scrum team makes an active commitment to upholding agile values, you'll have a successful agile transition.

## Signs Your Changes Are Slipping

The following list of questions helps you see warning signs and provide ideas on what to do if problematic circumstances arise:

### » Are you doing “scrum, but ...”?

“ScrumBut occurs when organizations partially adopt scrum. Some agile purists say that ScrumBut is unacceptable; other agile practitioners allow room for gradual growth into a new method. Having said that, beware of old practices that thwart agile principles, such as finishing sprints with incomplete functionality.



**REMEMBER** Scrum is three roles, three artifacts, and five events. If you find

your team tweaking those basic framework components, ask why. Is scrum exposing something you're not willing to inspect and adapt?

» **Are you still documenting and reporting in the old way?**

If you're still burning hours on hefty documentation and reporting, it's a sign that the organization has not accepted agile approaches for conveying project status. Help managers understand how to use existing agile reporting artifacts and quit doing double work!

» **A team completing 50 story points in a sprint is better than another team doing 10, right?**

No. Keep in mind that story points are relative and consistent within one scrum team, not across multiple scrum teams. Velocity isn't a team comparison metric. It is simply a post-sprint fact that scrum teams use to help them in their own planning. You can see more about story points and velocity in [Chapter 8](#).

» **When will the stakeholders sign off on all the specifications?**

If you're waiting for sign-offs on comprehensive requirements to start developing, you're not following agile practices. You can start development as soon as you have enough requirements for one sprint.

» **Are we using offshore to reduce costs?**

Ideally, scrum teams are collocated. The ability for instant face-to-face communication saves more time and money and prevents more costly mistakes than the initial hourly savings you may see with some offshore teams.

If you do work with offshore teams, invest in good collaboration tools such as individual video cameras and persistent, virtual team rooms.

» **Are development team members asking for more time in a sprint to finish tasks?**

The development team may not be working cross-functionally or swarming on priority requirements. Development team members can help one another finish tasks, even if those tasks are outside of a person's core expertise.

This question can also indicate outside pressures to underestimate tasks and fit more work into a sprint than the development team can handle.

» **Are development team members asking what they should do next?**

After a sprint is planned and development work is under way, if the developers are waiting for direction from the scrum master or product owner, they aren't self-organizing. The development team should be telling the scrum master and the product owner what it's doing next, not the other way round.

» **Are team members waiting until the end of the sprint to do testing?**

Agile development teams should test every day in a sprint. All development team members are testers.

» **Are the stakeholders showing up for sprint reviews?**

If the only people at sprint reviews are the scrum team members, it's time to remind stakeholders of the value of frequent feedback loops. Let stakeholders know that they're missing their chance to review working product functionality regularly, correct course early, and see firsthand how the project is progressing.

» **Is the scrum team complaining about being bossed around by the scrum master?**

Command-and-control techniques are the antithesis of self-management and are in direct conflict with agile principles. Scrum teams are teams of peers — the only boss is the team. Have a discussion with the agile mentor and act quickly to reset the scrum master's expectations of his or her role.

» **Is the scrum team putting in a lot of overtime?**

If the end of each sprint becomes a rush to complete tasks, you aren't practicing sustainable development. Look for root causes, such as pressure to underestimate. The scrum master may need to coach the development team and shield its members from product owner pressure if this is the case. Reduce the story points for each sprint until the development team can get a handle on the work.

» **What retrospective?**

If scrum team members start avoiding or cancelling sprint retrospectives, you're on the slide back to waterfall. Remember the importance of inspecting and adapting and be sure to look at why people are missing the retrospective in the first place. If you're not progressing, complacency usually results in sliding backwards. Even if the scrum team has great

velocity, development speed can always be better, so keep the retrospective, and keep improving.

## Part 6

# The Part of Tens

## **IN THIS PART ...**

Communicate the benefits of agile project management.

Address key factors for agile project success.

Measure progress in appropriately inspecting and adapting to become more agile as an organization.

Become an agile professional by learning, with support from valuable resources.

## Chapter 19

# Ten Key Benefits of Agile Project Management

---

### IN THIS CHAPTER

- » Ensuring that projects are rewarding
- » Making reporting easy
- » Improving results
- » Reducing risk

In this chapter, we provide ten important benefits that agile project management provides to organizations, project teams, and products.



REMEMBER To take advantage of agile project management benefits, you need to trust in agile practices, learn more about different agile approaches, and use what's best for your project team.

## *Better Product Quality*

Projects exist to build great products and purpose-driven outcomes. Agile methods have excellent safeguards to make sure that quality is as high as possible. Agile project teams help ensure quality by doing the following:

- » Take a proactive approach to quality to prevent product problems.
- » Embrace technological excellence, good design, and sustainable development.
- » Define and elaborate requirements just in time so that knowledge of product features is as relevant as possible.
- » Build acceptance criteria into user stories so that the development team

better understands them and the product owner can accurately validate them.

- » Incorporate continuous integration and daily testing into the development process, allowing the development team to address issues while they're fresh.
- » Take advantage of automated testing tools to develop during the day, test overnight, and fix defects in the morning.
- » Conduct sprint retrospectives, allowing the scrum team to continuously improve processes and work.
- » Complete work using the definition of done: developed, tested, integrated, and documented.

You can find more information about project quality in [Chapter 15](#).

## ***Higher Customer Satisfaction***

Agile project teams are committed to producing products that satisfy customers. Agile approaches for happier project sponsors include the following:

- » Collaborate with customers as partners and keep customers involved and engaged throughout projects.
- » Have a product owner who is an expert on product requirements and customer needs. (Check out [Chapters 6](#) and [9](#) to find out more information about the product owner role.)
- » Keep the product backlog updated and prioritized to respond quickly to change. (You can find out about the product backlog in [Chapter 8](#) and its role in responding to change in [Chapter 12](#).)
- » Demonstrate working functionality to customers in every sprint review. ([Chapter 10](#) shows you how to conduct a sprint review.)
- » Deliver products to market quicker and more often with every release.
- » Possess the potential for self-funding projects. ([Chapter 13](#) tells you about self-funding projects.)

# **Reduced Risk**

Agile project management techniques virtually eliminate the chance of absolute project failure — spending large amounts of time and money with no return on investment. Agile project teams run projects with lower risk by doing the following:

- » Develop in sprints, ensuring a short time between initial project investment and either failing fast or knowing that a product or an approach will work.
- » Always have a working, integrated product, starting with the first sprint, so that some value is added as shippable functionality every sprint, ensuring an agile project won't fail completely.
- » Develop requirements according to the definition of done in each sprint so that project sponsors have completed, usable functionality, regardless of what may happen with the project in the future.
- » Provide constant feedback on products and processes through the following:
  - Daily scrum meetings and constant development team communication
  - Regular daily clarification about requirements and review and acceptance of features by the product owner
  - Sprint reviews, with stakeholder and customer input about completed product functionality
  - Sprint retrospectives, where the development team discusses process improvement
  - Releases, where the end user can see and react to new features on a regular basis
- » Generate revenue early with self-funding projects, allowing organizations to pay for a project with little up-front expense.

You can find more information about managing risk in [Chapter 15](#).

# ***Increased Collaboration and Ownership***

When development teams take responsibility for projects and products, they can produce great results. Agile development teams collaborate and take ownership of product quality and project performance by doing the following:

- » Make sure that the development team, the product owner, and the scrum master work closely together on a daily basis.
- » Conduct goal-driven sprint planning meetings, allowing the development team to commit to the sprint goal and organize its work to achieve it.
- » Hold daily scrum meetings led by the development team, where development team members organize around work completed, future work, and roadblocks.
- » Conduct sprint reviews, where the development team can demonstrate and discuss the product directly with stakeholders.
- » Conduct sprint retrospectives, allowing development team members to review past work and recommend better practices with every sprint.
- » Work in a collocated environment, allowing for instant communication and collaboration among development team members.
- » Make decisions by consensus, using techniques such as estimation poker and the fist of five.

You can find out how development teams estimate effort for requirements, decompose requirements, and gain team consensus in [Chapter 7](#). To discover more about sprint planning and daily scrum meetings, see [Chapter 9](#). For more information about sprint retrospectives, check out [Chapter 10](#).

## ***More Relevant Metrics***

The metrics that agile project teams use to estimate time and cost, measure project performance, and make project decisions are often more relevant and more accurate than metrics on traditional projects. Agile metrics should

encourage sustainable team progress and efficiency in a way that works best for the team to deliver value to the customer early and often. On agile projects, you provide metrics by doing the following:

- » Determine project timelines and budgets based on each development team's actual performance and capabilities.
- » Make sure that the development team that will be doing the work, and no one else, provides effort estimates for project requirements.
- » Use relative estimates, rather than hours or days, to accurately tailor estimated effort to an individual development team's knowledge and capabilities.
- » Refine estimated effort, time, and cost on a regular basis, as the development team learns more about the project.
- » Update the sprint burndown chart every day to provide accurate metrics about how the development team is performing within each sprint.
- » Compare the cost of future development with the value of that future development, which helps project teams determine when to end a project and redeploy capital to a new project.



**WARNING** You might notice that velocity is missing from this list. *Velocity* (a measure of development speed, as detailed in [Chapter 13](#)) is a tool you can use to determine timelines and costs, but it works only when tailored to an individual team. The velocity of Team A has no bearing on the velocity of Team B. Also, velocity is great for measurement and trending, but it doesn't work as a control mechanism. Trying to make a development team meet a certain velocity number only disrupts team performance and thwarts self-management.

If you're interested in finding out more about relative estimating, be sure to check out [Chapter 7](#). You can find out about tools for determining timelines and budgets, along with information about capital redeployment, in [Chapter 13](#). [Chapter 21](#) shows you ten key metrics for agile project management.

## ***Improved Performance Visibility***

On agile projects, every member of the project team has the opportunity to know how the project is going at any given time. Agile projects can provide a high level of performance visibility by doing the following:

- » Place a high value on open, honest communication among the scrum team, stakeholders, customers, and anyone else in an organization who wants to know about a project.
- » Provide daily measurements of sprint performance with sprint backlog updates. Sprint backlogs can be available for anyone in an organization to review.
- » Provide daily insight into the development team's immediate progress and roadblocks through the daily scrum meeting. Although only the development team may speak at the daily scrum meeting, any member of the project team may attend.
- » Physically display progress by using task boards and posting sprint burndown charts in the development team's work area every day.
- » Demonstrate accomplishments in sprint reviews. Anyone within an organization may attend a sprint review.

Improved project visibility can lead to greater project control and predictability, as described in the following sections.

## ***Increased Project Control***

Agile project teams have numerous opportunities to control project performance and make corrections as needed because of the following:

- » Adjusting priorities throughout the project allows the organization to have fixed-time and fixed-price projects while accommodating change.
- » Embracing change allows the project team to react to outside factors such as market demand.
- » Daily scrum meetings allow the scrum team to quickly address issues as they arise.

- » Daily updates to sprint backlogs mean sprint burndown charts accurately reflect sprint performance, giving the scrum team the opportunity to make changes the moment it sees problems.
- » Face-to-face conversations remove roadblocks to communication and issue resolution.
- » Sprint reviews let project stakeholders see working products and provide input about the products before release.
- » Sprint retrospectives enable the scrum team to make informed course adjustments at the end of every sprint to enhance product quality, increase development team performance, and refine project processes.

The many opportunities to inspect and adapt throughout agile projects allow all members of the project team — the product owner, development team, scrum master, and stakeholders — to exercise control and ultimately create better products.

## ***Improved Project Predictability***

Agile project management techniques help the project team accurately predict how things will go as the project progresses. Here are some practices, artifacts, and tools for improved predictability:

- » Keeping sprint lengths and development team allocation the same throughout the project allows the project team to know the exact cost for each sprint.
- » Using individual development team speed allows the project team to predict timelines and budgets for releases, the remaining product backlog, or any group of requirements.
- » Using the information from daily scrum meetings, sprint burndown charts, and task boards allows the project team to predict performance for individual sprints.

You can find more information about sprint lengths in [Chapter 8](#).

## ***Customized Team Structures***

Self-management puts decisions that would normally be made by a manager or the organization into scrum team members' hands. Because of the limited size of development teams — which consist of three to nine people — agile projects can have multiple scrum teams on one project if necessary. Self-management and size-limiting mean that agile projects can provide unique opportunities to customize team structures and work environments. Here are a few examples:

- » Development teams may organize themselves into groups with specific skills or that work on specific parts of the product system and features.
- » Development teams may organize their team structure around people with specific work styles and personalities. Organization around work styles provides these benefits:
  - Allows team members to work the way they want to work
  - Encourages team members to expand their skills to fit into teams they like
  - Helps increase team performance because people who do good work like to work together and naturally gravitate toward one another
- » Scrum teams can make decisions tailored to provide balance between team members' professional and personal lives.
- » Ultimately, scrum teams can make their own rules about whom they work with and how they work.



**REMEMBER** The idea of team customization allows agile workplaces to have more diversity. Organizations with traditional management styles tend to have monolithic teams where everyone follows the same rules. Agile work environments are much like the old salad bowl analogy. Just like salads can have ingredients with wildly different tastes that fit in to make a delicious dish, agile projects can have people on teams with very diverse strengths that fit in to make great products.

# **Higher Team Morale**

Working with happy people who enjoy their jobs can be satisfying and rewarding. Agile project management improves the morale of scrum teams in these ways:

- » Being part of a self-managing team allows people to be creative, innovative, and acknowledged for their contributions.
- » Focusing on sustainable work practices ensures that people don't burn out from stress or overwork.
- » Encouraging a servant-leader approach assists scrum teams in self-management and actively avoids command-and-control methods.
- » Having a dedicated scrum master, who serves the scrum team, removes impediments, and shields the development team from external interferences.
- » Providing an environment of support and trust increases people's overall motivation and morale.
- » Having face-to-face conversations helps reduce the frustration of miscommunication.
- » Working cross-functionally allows development team members to learn new skills and to grow by teaching others.

You can find out more about team dynamics in [Chapter 14](#).

# Chapter 20

## Ten Key Factors for Project Success

---

### IN THIS CHAPTER

- » Ensuring scrum teams have the environment and tools they need
- » Filling all roles with the right talent
- » Enabling teams with clear direction and support

Here are ten key factors that determine whether an agile transition will succeed. You don't need all issues resolved before you begin. You just need to be aware of them and have a plan to address them as early in your journey as possible.



TIP We have found that the first three are the strongest indicators for success. Get those right and the likelihood of your success increases dramatically.

### *Dedicated Team Members*

In [Chapter 6](#), we talk about the importance of dedicating team members — product owner, development team members, as well as scrum master — to a single project at a time. This is especially critical at the beginning, when the scrum team and the rest of the organization are still learning what it means to value agility and embody agile principles.

If team members are jumping between project contexts hourly, daily, weekly, or even monthly, the focus on getting agile techniques right is minimized at the expense of just trying to keep up with multiple task lists. Also, the time lost due to the continual cognitive demobilization and remobilization

involved with task switching is very costly to each project involved.



TIP If you think you don't have enough people to dedicate to your scrum teams, you definitely don't have enough people to thrash them across multiple projects simultaneously. The American Psychological Association reports that task switching wastes as much as 40 percent of time.

## Collocation

The Agile Manifesto lists individuals and interactions as the first value. The way you get this value right is by collocating team members to be able to have clear, effective, and direct communication throughout a project.

In [Chapter 5](#), we talk about collocation as the first crucial element of an agile environment. Bell Laboratories showed a fifty-fold improvement in productivity simply by getting individuals and interactions right through collocation. With this success factor addressed adequately, customer collaboration, working functionality, and responding effectively to change become much more of a reality.

## Automated Testing

Development teams cannot develop at the rate technology and market conditions change if they have to manually test their work every time they integrate new pieces of functionality throughout the sprint. The longer teams rely on manual testing, the larger the holes in test coverage become — manual testing simply takes too long and in reality becomes spot-checking. Without automation, scrum teams will struggle to completely deliver value in every sprint.

In [Chapter 4](#), we discuss extreme programming practices aimed at building in quality upfront; automated testing is one of the primary practices. [Chapter 15](#) also discusses building in quality through automation and continuous integration.

# ***Enforced Definition of Done***

Ending sprints with non-shippable functionality is an anti-pattern to becoming more agile. Your definition of done should clarify the following:

- » The environment in which the functionality should be integrated
- » The types of testing
- » The types of required documentation

The scrum team should also enforce its definition of done. If scrum teams tell their stakeholders that they are done after a sprint, but an aspect of the definition of done is not met, the work to finish meeting the definition of done must be added to the next sprint, taking capacity away from working on new valuable product backlog items. This scenario is a Ponzi scheme.

Development teams get to done by swarming on user stories — working on one user story together at a time until it is complete before starting the next. Developers hold each other accountable by ensuring that all rules for their definition of done are satisfied before starting a new user story. Product owners review completed work against the scrum team's definition of done (as well as the user story's acceptance criteria — see [Chapter 8](#)) as soon as developers complete it, and the scrum master ensures that developers resolve issues rejected by the product owner before moving on to new user stories.

Swarming to follow a clear definition of done makes sprints successful. See [Chapters 2, 8, 10](#), and [15](#) for more on the definition of done.

# ***Clear Product Vision and Roadmap***

Although the product owner owns the product vision and product roadmap, many people have the responsibility to ensure the clarity of these agile artifacts. Product owners need access to stakeholders and customers at the beginning during project planning as well as throughout the project to ensure that the vision and roadmap continually reflect what the customer and market need. Purpose-driven development delivers business and customer value and mitigates risk effectively.

Without a clear purpose, people wander and lack ownership. When all team

members understand the purpose, they come together. Remember the agile principle, “The best architectures, requirements, and designs emerge from self-organizing teams.”

We discuss the mechanics of developing the vision and product roadmap in [Chapter 7](#).

## ***Product Owner Empowerment***

The product owner’s role is to optimize the value produced by the development team. This product owner responsibility requires someone to be knowledgeable about the product and customer, available to the development team throughout each day, and empowered to make priority decisions and give clarification in the moment so that development teams don’t wait or make inappropriate decisions for the product’s direction.

Although all roles on the scrum team are vital and equally important, an unempowered and ineffective product owner usually causes scrum teams to ultimately fail at delivering the value customers need from the team. See [Chapter 6](#) for more on the product owner role.

## ***Developer Versatility***

You probably won’t start your first agile project with a development team that has the ideal level of skills required for every requirement on your product backlog. However, the goal should be to achieve skill coverage as soon as possible. Your team will also be challenged to meet its sprint goal if you have single points of failure on any one skill, including testing.

From day one, you need developers on your team with the intellectual curiosity and interest to learn new things, to experiment, to mentor and to receive mentoring, and to work together as a team to get things to done as quickly as possible. This versatility is discussed more in [Chapter 6](#).

## ***Scrum Master Clout***

As you depart from command and control leadership to empower the people doing the work to make decisions, servant leadership provides the solution.

With formal authority, a scrum master would be viewed as a manager — someone to report to. Scrum masters should not be given formal authority but should be empowered by leadership to work with members of the scrum team, stakeholders, and other third parties to clear the way so that the development team can function unhindered.

If scrum masters have organizational clout, which is informal and is a socially earned ability to influence, they can best serve their teams to optimize their working environment. In [Chapter 6](#), we talk more about different types of clout. Provide training and mentorship to ensure that your scrum masters develop the soft skills of servant leadership and put off the tendencies of commanding and directing.

## ***Management Support for Learning***

When executive leaders decide to become agile, their mindset has to change. Too often we see leadership directives without any follow-through for supporting the learning process to implement the changes. It is unrealistic to expect all the benefits of following agile principles after the first sprint. In [Chapter 18](#), we talk about choosing an appropriate agile pilot project, one with leeway to stumble a bit at first as everyone learns a new way of working together.

The bottom line: If support for learning is merely lip service, scrum teams will pick up on it early, will lose motivation to try new things, and will go back to waiting for top-down directives on how to do their job.

## ***Transition Support***

[Chapter 18](#) compares an agile transition to a sports team transitioning to play a different sport. Good coaching at leadership and team levels increases your chances to succeed. Coaching provides support in the following forms:

- » In-the-moment course correction when discipline starts to slip or mistakes are made
- » Reenforcing training
- » One-on-one mentoring for specific role-based challenges

» Executive leadership style and mindset adjustments

See our Platinum Edge agile transition roadmap in [Chapter 18](#) for specific steps to take alongside your trusted agile expert coaches.

# Chapter 21

## Ten Metrics for Agile Organizations

---

### IN THIS CHAPTER

- » Using success metrics
- » Calculating time and cost metrics
- » Understanding satisfaction metrics

On an agile project, metrics can be powerful tools for planning, inspecting, adapting, and understanding progress over time. Rates of success or failure can let a scrum team know whether it needs to make positive changes or keep up its good work. Time and cost numbers can highlight the benefits of agile projects and provide support for an organization's financial activities. Metrics that quantify people's satisfaction can help a scrum team identify areas for improvement with customers and with the team itself.

This chapter describes ten key metrics to help guide agile project teams.

## *Return on Investment*

*Return on investment (ROI)* is income generated by the product less project costs: money in versus money out. ROI is fundamentally different in agile projects than it is in traditional projects. Agile projects have the potential to generate income with the first release and can increase revenue with each new release.

To fully appreciate the difference between ROI on traditional and agile projects, compare the examples in [Tables 21-1](#) and [21-2](#). The projects for both examples have the same project costs and take the same amount of time to complete. Both products have the potential to generate \$100,000 in income every month when all the requirements are finished.

**TABLE 21-1** ROI on a Traditional Project

<b>Month</b>	<b>Monthly Income</b>	<b>Monthly Costs</b>	<b>Monthly ROI</b>	<b>Total Income</b>	<b>Total Costs</b>	<b>Total ROI</b>
January	\$0	\$80,000	-\$80,000	\$0	\$80,000	-\$80,000
February	\$0	\$80,000	-\$80,000	\$0	\$160,000	-\$160,000
March	\$0	\$80,000	-\$80,000	\$0	\$240,000	-\$240,000
April	\$0	\$80,000	-\$80,000	\$0	\$320,000	-\$320,000
May	\$0	\$80,000	-\$80,000	\$0	\$400,000	-\$400,000
June (project launch)	\$0	\$80,000	-\$80,000	\$0	\$480,000	-\$480,000
July	\$100,000	\$0	\$100,000	\$100,000	\$480,000	-\$380,000
August	\$100,000	\$0	\$100,000	\$200,000	\$480,000	-\$280,000
September	\$100,000	\$0	\$100,000	\$300,000	\$480,000	-\$180,000
October	\$100,000	\$0	\$100,000	\$400,000	\$480,000	-\$80,000
November (breakeven)	\$100,000	\$0	\$100,000	\$500,000	\$480,000	\$20,000
December	\$100,000	\$0	\$100,000	\$600,000	\$480,000	\$120,000

**TABLE 21-2** ROI on an Agile Project

<b>Month</b>	<b>Monthly Income</b>	<b>Monthly Costs</b>	<b>Monthly ROI</b>	<b>Total Income</b>	<b>Total Costs</b>	<b>Total ROI</b>
January	\$0	\$80,000	-\$80,000	\$0	\$80,000	-\$80,000
February	\$15,000	\$80,000	-\$65,000	\$15,000	\$160,000	-\$145,000
March	\$25,000	\$80,000	-\$55,000	\$40,000	\$240,000	-\$200,000
April	\$40,000	\$80,000	-\$40,000	\$80,000	\$320,000	-\$240,000
May	\$70,000	\$80,000	-\$10,000	\$150,000	\$400,000	-\$250,000
June (project end)	\$80,000	\$80,000	\$0	\$230,000	\$480,000	-\$250,000
July	\$100,000	\$0	\$100,000	\$330,000	\$480,000	-\$150,000

							\$150,000
August	\$100,000	\$0	\$100,000	\$430,000	\$480,000	-\$50,000	
September (breakeven)	\$100,000	\$0	\$100,000	\$530,000	\$480,000	\$50,000	
October	\$100,000	\$0	\$100,000	\$630,000	\$480,000	\$150,000	
November	\$100,000	\$0	\$100,000	\$730,000	\$480,000	\$250,000	
December	\$100,000	\$0	\$100,000	\$830,000	\$480,000	\$350,000	

First, look at the ROI on a traditional project in [Table 21-1](#).

Here are some key points of the traditional project in [Table 21-1](#):

- » The project first generated income in July, after the project launch the end of June.
- » The project finally had a positive total ROI in November, 11 months after the project started.
- » By the end of one year, the project generated \$600,000 in revenue.
- » At the year's end, the project's total ROI was \$120,000.

Now look at the ROI for an agile project in [Table 21-2](#).

Pay special attention to these points of the agile project in [Table 21-2](#):

- » The project first generated income in February, shortly after the project start.
- » The project had a positive total ROI in September — two months earlier than the traditional project.
- » By the end of one year, the project generated \$830,000 in revenue, nearly 40 percent more than the traditional project.
- » At the year's end, the total ROI was \$350,000, almost threefold the ROI on the traditional project.



**REMEMBER** Like time to market, ROI metrics are a great way for an organization to appreciate the ongoing value of an agile project. ROI metrics help

justify projects from the start because companies can fund projects based on ROI potential. Organizations can track ROI for individual projects as well as for the organization as a whole.

## New requests in ROI budgets

An agile project's capability to quickly generate high ROI provides organizations with a unique way to fund additional product development. New product functionality may translate to higher product income.

For example, suppose that in the example project from [Table 21-2](#), the project team were to identify a new feature that would take one month to complete and would boost the product income from \$100,000 a month to \$120,000 a month. Here's what the effect would be on ROI:

- » The project would still have its first positive ROI in September, with an ROI of \$110,000 instead of \$50,000.
- » By the end of the year, the project would have generated a total income of \$950,000 — 14 percent more than if it generated \$100,000 a month.
- » By the end of the year, the total ROI would be \$470,000 — 34 percent higher than the original project.

If a project is already generating income, it can make sense for an organization to roll that income back into new development and see higher revenue.

## Capital redeployment

On an agile project, when the cost of future development is higher than the value of that future development, it's time for the project to end.

The product owner prioritizes requirements, in part, by their capability to generate revenue or value. If only low-revenue or low-value requirements remain in the backlog, a project may end before the project team has used its entire budget. The organization may then use the remaining budget from the old project to start a new, more valuable project. The practice of moving a budget from one project to another is called *capital redeployment*.

To determine a project's end, you need the following metrics:

- » The value (V) of the remaining requirements in the product backlog

- » The actual cost (AC) for the work to complete the requirements in the product backlog
- » The opportunity cost (OC), or the value of having the scrum team work on a new project

When  $V < AC + OC$ , the project can stop. The cost you would sink into the project would be more than the value you would receive from the project.

Capital redeployment allows an organization to spend efficiently on valuable product development and maximize the organization's overall ROI. You can find the details on capital redeployment in [Chapter 13](#).

## **Satisfaction Surveys**

On agile projects, a scrum team's highest priority is to satisfy the customer early and often. At the same time, the scrum team strives to motivate individual team members and promote sustainable development practices.

A scrum team can benefit from digging deeper into customer and team member experiences. One way to get measurable information about how well a scrum team is embodying agile principles is through satisfaction surveys:

- » **Customer satisfaction surveys:** Measure the customer's experience with the project, the process, and the scrum team.

The scrum team may want to use customer surveys multiple times during a project. The scrum team can use customer survey results to examine processes, continue positive practices, and adjust behavior as necessary.

- » **Team satisfaction surveys:** Measure the scrum team members' experience with the organization, the work environment, processes, other project team members, and their work. Everyone on the scrum team can take team surveys.

As with the customer survey, the scrum team may choose to give team surveys throughout a project. Scrum team members can use team survey results to regularly fine-tune and adjust personal and team behaviors. The scrum team can also use results to address organizational issues.

Customer survey results over time can provide a quantitative look at how the scrum team is maturing as a team.



TIP Survey results will be more honest and freely given if the organization fosters a culture of openness, transparency, and support for learning.

You can put together informal paper surveys, or use one of the many online survey tools. Some companies even have survey software available through their human resources department.

## Defects in Production

Defects are a part of any project. However, testing and fixing them can be time-consuming and costly, especially when they reach production. Agile approaches help development teams proactively minimize defects.

As development teams iterate through the development of a requirement, they test and find defects. The sprint cycle facilitates fixing those defects immediately before they reach production. Ideally, defects in production don't occur due to automated testing and continuous integration, as discussed in [Chapters 7](#) and [15](#).

Tracking defect metrics can let the development team know how well it's preventing issues and when to refine its processes. To track defects, it helps to look at the following numbers:

- » **Build defects:** If the development team uses automated testing and continuous integration, it can track the number of defects at the build level in each sprint.

By understanding the number of build defects, the development team can know whether to adjust development processes and environmental factors to be able to catch defects even sooner in their development process.

- » **User acceptance testing (UAT) defects:** The development team can track the number of defects the product owner finds when reviewing completed functionality in each sprint.

By tracking UAT defects, the development team and the product owner can identify the need to refine processes for understanding requirements. The development team can also determine whether adjustments to

automated testing tools are necessary.

- » **Release defects:** The development team can track the number of defects that make it past the release to the marketplace.



**TIP** Development teams can also track the number of days between user story acceptance and defect discovery. The fewer days passed since a developer worked on the functionality, the lower the cost to fix the defect.

By tracking release defects, the development team and the product owner can know whether changes to the UAT process, automated testing, or the development process are necessary. Large numbers of defects at the release level may indicate bigger problems in the scrum team.

The number of defects and whether defects are increasing, decreasing, or staying the same are good metrics to spark discussions on project processes and development techniques at sprint retrospectives.

You can find out more about proactive quality management and testing in [Chapter 15](#).

## **Sprint Goal Success Rates**

One way to measure agile project performance is the rate of achieving the sprint goal.



**REMEMBER** The sprint may not need all the requirements and tasks in the sprint backlog to achieve the goal. However, a successful sprint should have a working product increment that fulfills the sprint goals and meets the scrum team's definition of done: developed, tested, integrated, and documented.

Throughout the project, the scrum team can track how frequently it succeeds in reaching the sprint goals and use success rates to see whether the team is maturing or needs to correct its course. Sprint success rates are a useful launching point for inspection and adaptation.

You can find out more about setting sprint goals in [Chapter 8](#).

## Time to Market

*Time to market* is the amount of time an agile project takes to provide value by releasing working functionality to users. Organizations may perceive value in a couple of ways:

- » When a product directly generates income, its value is the money it can make.
- » When a product is for an organization's internal use, its value will be the employees' ability to use the product and will contain subjective factors based on what the product can do.

When measuring time to market, consider the following:

- » Measure the time from the project start until you first show value.
- » Some scrum teams deploy new product features for use at the end of each sprint. For scrum teams that release with every sprint, the time to market is simply the sprint length, measured in days.
- » Other scrum teams plan releases after multiple sprints and deploy product features in groups. For scrum teams that use longer release times, the time to market is the number of days between each release.

Time to market helps organizations recognize and quantify the ongoing value of agile projects. Time to market is especially important for companies with revenue-generating products because it aids in budgeting throughout the year. It's important also if you have a self-funding project — a project being paid for by the income from the product.

You can find out more about product-income generation and self-funding projects in [Chapter 13](#).

## Lead and Cycle Times

*Lead time* is the average amount of time between receiving a request for a requirement and delivering it finished. *Cycle time* is the average time between

when development on a requirement begins and when it is delivered.

Agile teams work in a lean environment, one that seeks to eliminate waste. Constraints exist in every flow of work or stream of creating value. Agile teams continually seek ways to identify and remove constraints to maximize the flow of work through their system.

Lead and cycle time provide not only a measurement of where bottlenecks may exist but also expectations for stakeholders regarding how long a request they submit may take to be completed, on average.

If the lead time for a particular scrum team is 45 days but the cycle time is only 5 days, this discrepancy may alert the team to evaluate its planning and product backlog refinement process to see how it might tighten the 40-day difference. Likewise, if the lead time is 45 days and the cycle time is 40 days, the team may want to evaluate its development workflow for bottlenecks. In any case, scrum teams should always be looking at removing constraints to decrease both lead and cycle time appropriately.

## ***Cost of Change***

Agile leaders and teams embrace change for the customer's competitive advantage. But acceptance of change should not mean acceptance of unnecessary costs associated with changes. As agile teams inspect and adapt the product and their processes, their goal should be to continuously minimize the effect of change.

Increasing product flexibility is a common way of reducing the cost of change. With software development, using a service oriented architecture (SOA) strategy allows agile teams to make each component of an application independent of others, so that the entire system doesn't require changes when one component must be changed. Development, testing, and documentation require significantly less effort.

In manufacturing, standardization and modularization of parts has allowed car manufacturers such as Toyota and more recently WikiSpeed to build cars more quickly and with less wasteful rework due to incompatibility. (See [Chapter 4](#) for more on the Toyota Production System.) Value stream mapping is a common technique for identifying constraints in a system or a workflow. By visualizing (on a whiteboard, for instance) each step in a process, agile

teams can identify where introducing changes forces the most stress or cost on its processes. When a constraint is identified, the scrum master and other organizational change agents can work to remove that constraint to decrease the cost of future changes in the system.

## **Team Member Turnover**

Agile projects tend to have higher morale. One way of quantifying morale is by measuring turnover. Although turnover isn't always directly tied to satisfaction, it can help to look at the following metrics:

- » **Scrum team turnover:** Low scrum team turnover can be one sign of a healthy team environment. High scrum team turnover can indicate problems with the project, the organization, the work, individual scrum team members, burnout, ineffective product owners forcing development team commitments, personality incompatibility, a scrum master who fails to remove impediments, or overall team dynamics.
- » **Company turnover:** High company turnover, even if it doesn't include the scrum team, can affect morale and effectiveness. High company turnover can be a sign of problems in the organization. As a company adopts agile practices, it may see turnover decrease.

When the scrum team knows turnover metrics and understands the reasons behind those metrics, it may be able to take actions to maintain morale and improve the work environment. If turnover is high, start asking why.

## **Skill Versatility**

Mature scrum teams are typically more cross functional than less mature scrum teams. By eliminating single points of failure in a scrum team, you increase its ability to move faster and produce higher-quality products. Tracking skill versatility allows scrum teams and functional managers to gauge the growth of cross-functionality.

When starting out, capture the existing skills and levels contained at each of the following organizational structures:

- » Per person skills and levels
- » Per team skills and levels
- » Per organization skills and levels

Over time, as each person increases his or her quantity and level of skills, the constraints and delays due to skill gaps disappear. Agile teams are about skills, not titles. You want team members who can contribute to the sprint goal each and every day without the risk of single points of failure.

## ***Manager-to-Creator Ratio***

Larger organizations likely have developed a heavy middle layer of managers. Many organizations haven't figured out how to function well without multiple managers handling personnel, training, and technical direction on development issues. However, you need to strike the right balance of managers and individuals who produce product.

Imagine two professional rival futbol (American soccer) teams of 11 players each who both train intensively and prepare for a match against each other. Team B beats Team A 1-0.

Both teams go back to train for the next match. Team A's management calls on an analyst to provide a solution. After careful analysis of both teams, he sees that Team B has one player as goalkeeper, and ten spread across the field as defenders, midfielders, and forwards, while Team A plays ten goalkeepers at once and one forward to maneuver the ball down the field to the goal without any team members getting in the way.

Team A's management calls in a consultant to restructure the team. She finds what seems obvious: Team A is playing way too many goalkeepers. The consultant recommends that the team play half as many goalkeepers (five), and play five defenders who can relay instructions to the forward from the goalkeepers who have a view of the entire field. She also suggests doubling the assistant coaching staff to increase training and motivation of the forward to score goals.

At the next match, Team B again beats Team A, but this time 2-0.

The forward gets cut, the assistant coaches and defenders get recognized for

their motivation strategy, but management calls for another analysis. As a result of the analysis, they build a more modern practice facility and invest in the latest shoe technology for the next season.



**REMEMBER** Every dollar spent on someone who manages organizational processes is a dollar not spent on a product creator.

Track your manager-to-creator ratio to identify bloat and ways to minimize the investment you're making in people who don't create product.

## Chapter 22

# Ten Valuable Resources for Agile Professionals

---

### IN THIS CHAPTER

- » Finding support for successful agile transitions
- » Getting involved with active agile communities
- » Accessing resources for common agile approaches

Many organizations, websites, blogs, and companies exist to provide information about and support for agile project management. To help you get started, we compiled a list of ten resources that we think are valuable to support your journey to agile project management.

## *Agile Project Management For Dummies Online Cheat Sheet*

[www.dummies.com](http://www.dummies.com)

You can use our online cheat sheet as a companion to this book as you start implementing the agile values and principles from the Agile Manifesto, as well as models outlined throughout the book. You'll find how-to guides, tools, templates, and other helpful resources for your agile toolkit. To get to the cheat sheet, go to [www.dummies.com](http://www.dummies.com), and then type *Agile Project Management For Dummies* in the Search box.

## *Scrum For Dummies*

In 2014, we published *Scrum For Dummies* (Wiley) as a field guide not only to scrum but also to scrum in industries and business functions outside information technology (IT) and software development. Scrum can be applied

in any situation where you want early empirical feedback on what you're building or pursuing on a project.

Learn about scrum in industries such as game software development and tangible goods production (construction, manufacturing, hardware development) and in services such as healthcare, education, and publishing.

Explore scrum's applications in business functions, including operations, portfolio management, human resources, finance, sales, marketing, and customer service.

And in everyday life, see how scrum can help you organize your pursuits of dating, family life, retirement planning, and education.

## ***The Scrum Alliance***

[www.scrumalliance.org](http://www.scrumalliance.org)

The Scrum Alliance is a nonprofit professional membership organization that promotes the understanding and usage of scrum. The alliance achieves this goal by promoting scrum training and certification classes, hosting international and regional scrum gatherings, and supporting local scrum user communities. The Scrum Alliance site is rich in blog entries, white papers, case studies, and other tools for learning and working with scrum. [Chapter 16](#) lists many of the Scrum Alliance certifications.

## ***The Agile Alliance***

[www.agilealliance.org](http://www.agilealliance.org)

The Agile Alliance is the original global agile community, with a mission to help advance the 12 Agile Principles and common agile practices, regardless of approach. The Agile Alliance site has an extensive resources section that includes articles, videos, presentations, and an index of independent agile community groups across the world.

## ***The Project Management Institute Agile Community***

[www.projectmanagement.com/practices/agile](http://www.projectmanagement.com/practices/agile)

The Project Management Institute (PMI) is the largest nonprofit project management membership association in the world. It has more than 400,000 members and a presence in more than 200 countries. PMI supports an agile community of practice and an agile certification, the PMI Agile Certified Practitioner (PMI-ACP).

The PMI website provides information and requirements for certification along with access to papers, books, and seminars about agile project management. PMI members can also access PMI's agile community website, with an extensive knowledge center including blog posts, forums, webinars, and information about local agile networking events.

## ***International Consortium for Agile (ICAgile)***

[icagile.com](http://icagile.com)

ICAgile is a community-driven organization helping people become agile through education, awareness, and certification. Its learning roadmap provides career path development support in business agility, enterprise and team agile coaching, value management, delivery management, agile engineering, agile testing, and DevOps.

## ***InfoQ***

[www.infoq.com/agile](http://www.infoq.com/agile)

InfoQ is an independent online community with a prominent agile section offering news, articles, video interviews, video presentations, and minibooks, all written by domain experts in agile techniques. The resources at InfoQ tend to be high quality, and the content is both unique and relevant to the issues facing agile project teams.

## ***Lean Enterprise Institute***

[www.lean.org](http://www.lean.org)

Lean Enterprise Institute publishes books, blogs, knowledge bases, news, and events for the broader community of lean thinkers and practitioners. As you pursue agile project management, remember to incorporate lean thinking in all that you do. Lean.org is a good launching pad for you to explore the lean topics relevant to your situation.

## ***Extreme Programming***

[ronjeffries.com/xprog/what-is-extreme-programming/](http://ronjeffries.com/xprog/what-is-extreme-programming/)

Ron Jeffries was one of the originators of the extreme programming (XP) development approach, along with Kent Beck and Ward Cunningham. Ron provides resources and services in support of XP's advancement on his ronjeffries.com site. The "[What Is Extreme Programming?](#)" section of the site summarizes the core concepts of XP. Other articles and extreme programming resources are also available in wiki format at <http://wiki.c2.com/?ExtremeProgrammingCorePractices>.

## ***Platinum Edge***

[www.platinumedge.com](http://www.platinumedge.com)

Since 2001, our team at Platinum Edge has been helping companies maximize organizational return on investment (ROI). Visit our blog to get the latest insights on practices, tools, and innovative solutions emerging from our work with Global 1000 companies and the dynamic agile community.

We also provide the following services, which are outlined in more detail in [Chapter 18](#):

- » **Agile audits:** Auditing your current organizational structure and processes to create an agile implementation strategy that delivers bottom-line results.
- » **Recruiting:** With access to the best agile and scrum talent — because we've trained them — we help you find the right people to bootstrap your scrum projects, including scrum masters, scrum product owners, and scrum developers.
- » **Training:** Public and private customized corporate agile and scrum

training and certification, regardless of your level of knowledge. In addition to custom and non-certified training options, we offer the following certifications:

- Certified ScrumMaster classes (CSM)
- Certified Scrum Product Owner classes (CSPO)
- Certified Scrum Developer classes (CSD)
- SAFe Scaled Agile training and implementations
- PMI Agile Certified Practitioner (PMI-ACP) test preparation classes

» **Transformation:** Nothing is a larger factor of future success than proper coaching. We follow up on agile training with embedded agile coaching and mentoring to ensure that the right practices occur in the real world.

# About the Authors

**Mark C. Layton**, known globally as *Mr. Agile*, is an organizational strategist and Scrum Alliance certification instructor with over 20 years in the project/program management field. He is the Los Angeles chair for the Agile Leadership Network, the author of the international *Scrum For Dummies* and *Agile Project Management For Dummies* book series (both published by Wiley), the creator of the *Agile Foundations Complete Video Course* with Pearson Education, and the founder of Platinum Edge, LLC — an organizational improvement company that supports businesses making the waterfall-to-agile transition.

Prior to founding Platinum Edge in 2001, Mark developed his expertise as a consulting firm executive, a program management coach, and an in-the-trenches project leader. He also spent 11 years as a Cryptographic Specialist for the US Air Force, where he earned both Commendation and Achievement medals for his accomplishments.

Mark holds MBAs from the University of California, Los Angeles, and the National University of Singapore; a B.Sc. (*summa cum laude*) in Behavioral Science from Pitzer College/University of La Verne; and an A.S. in Electronic Systems from the Air Force's Air College. He is also a Distinguished Graduate of the Air Force's Leadership School, a Certified Scrum Trainer (CST), a certified Project Management Professional (PMP), a recipient of Stanford University's advanced project management certification (SCPM), and a certified Scaled Agile Framework Program Consultant (SAFe SPC).

In addition to his books and videos, Mark is a frequent speaker at major conferences on Lean, Scrum, XP, and other agile solutions.

Additional information can be found at [platinumedge.com](http://platinumedge.com).

**Steven J Ostermiller** is a coach, mentor, and trainer empowering leaders, teams, and individuals to become more agile. Steve is co-founder and organizer of Utah Agile (sponsored by Agile Alliance, Scrum Alliance, and Agile Leadership Network), a professional community committed to increasing agility for Utah businesses, technology, and individuals. He developed and taught a business college's agile project management

curriculum and serves on its project management advisory board. Steve was also technical editor on projects such as *Scrum For Dummies* and Pearson Education's *Agile Foundations Complete Video Course*. He also occasionally speaks and writes about his experience with agile techniques for households.

Steve facilitates Platinum Edge, LLC's agile transformation engagements through audit, recruiting, training, and embedded coaching. He has worked with executive leadership and individual teams in finance, healthcare, media, entertainment, defense and energy, local and state government, logistics, ecommerce, manufacturing, ERP implementations, PMO development, startups, and nonprofits. He is a Certified Scrum Professional (CSP) and a Project Management Professional (PMP), and holds a B.S. in Business Management/Organizational Behavior from the Marriott School of Management at Brigham Young University.

Steve spends as much time as possible with his adorable wife and their five charming children in Utah living their dreams, one home-cooked meal at a time.

## **Dedications**

To the friends, family, and special loved ones who tirelessly love and support as I pursue these wild ideas. Your time is now. — Mark

To Gwen, my complete and final answer. And to our five littles, who give me every reason to continuously inspect and adapt. — Steve

## ***Authors' Acknowledgments***

We'd like to again thank the numerous people who contributed to the first edition of this book and helped make it a reality.

We are also very grateful to those who helped make this second edition a more valuable field guide: David Morrow for his insight and technical editing; Caroline Patchen for ensuring that these concepts are more easily understood through clear visualization; Jeff Sutherland, Ken Schwaber, Kurt Bittner, Patricia Kong, Dean Leffingwell, Alex Yakyma, Inbar Oren, Craig Larman, Bas Vodde, Mike Beedle, and Michael Herman for providing scaling options to the public and for their valuable feedback with the new scaling chapter; and to Amy Fandrei, Susan Pink, and the broader John Wiley & Sons team. You are all fantastic professionals; thank you for the opportunity to make this book even better.

And a shout-out to the signers of the Agile Manifesto. Thanks for coming together, finding common ground, and kickstarting the discussion that inspires us to keep becoming more agile.

## **Publisher's Acknowledgments**

**Acquisitions Editor:** Amy Fandrei

**Project Editor:** Susan Pink

**Copy Editor:** Susan Pink

**Technical Editor:** David Morrow

**Sr. Editorial Assistant:** Cherie Case

**Production Editor:** Antony Sami

**Cover Image:** © wsfurlan/iStockphoto

# Take Dummies with you everywhere you go!



Go to our [Website](#)



Like us on [Facebook](#)



Follow us on [Twitter](#)



Watch us on [YouTube](#)



Join us on [LinkedIn](#)



Pin us on [Pinterest](#)



Circle us on [google+](#)



Subscribe to our [newsletter](#)



Create your own [Dummies book cover](#)



[Shop Online](#)

FOR  
**DUMMIES**  
A Wiley Brand

# **WILEY END USER LICENSE AGREEMENT**

---

Go to [www.wiley.com/go/eula](http://www.wiley.com/go/eula) to access Wiley's ebook EULA.