

LEARNING MADE EASY



3rd Edition

Agile Project Management

for
dummies[®]

A Wiley Brand



Make performance
be progress

—
Deliver customer value
in weeks, not months

—
Turn agile principles
into practice

Mark C. Layton

CST, PMP, MBA²

Steven J Ostermiller

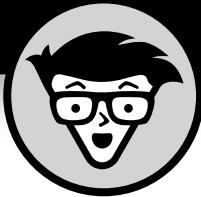
CST, PMP, ICP-ACC

Dean J. Kynaston

CSP-SM, CSP-PO, MBA

Agile Project Management

for
dummies[®]
A Wiley Brand



Agile Project Management

3rd Edition

by **Mark C. Layton, Steven J Ostermiller,
and Dean J. Kynaston**

for
dummies[®]
A Wiley Brand

Agile Project Management For Dummies®, 3rd Edition

Published by: **John Wiley & Sons, Inc.**, 111 River Street, Hoboken, NJ 07030-5774, www.wiley.com

Copyright © 2020 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.

Library of Congress Control Number: 20200942690

ISBN 978-1-119-67699-7 (pbk); ISBN 978-1-119-67706-2 (ebk); ISBN 978-1-119-67705-5 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Contents at a Glance

Introduction	1
Part 1: Understanding Agility	5
CHAPTER 1: Modernizing Project Management	7
CHAPTER 2: Applying the Agile Manifesto and Principles	21
CHAPTER 3: Why Being Agile Works Better	49
CHAPTER 4: Agility Is about Being Customer Focused	69
Part 2: Being Agile	89
CHAPTER 5: Agile Approaches	91
CHAPTER 6: Agile Environments in Action	109
CHAPTER 7: Agile Behaviors in Action	123
CHAPTER 8: The Permanent Team.....	149
Part 3: Agile Planning and Execution.....	159
CHAPTER 9: Defining the Product Vision and Product Roadmap.....	161
CHAPTER 10: Planning Releases and Sprints	183
CHAPTER 11: Working throughout the Day	215
CHAPTER 12: Showcasing Work, Inspecting, and Adapting	239
Part 4: Agility Management.....	251
CHAPTER 13: Managing a Portfolio: Pursuing Value over Requirements	253
CHAPTER 14: Managing Scope and Procurement.....	269
CHAPTER 15: Managing Time and Cost.....	287
CHAPTER 16: Managing Team Dynamics and Communication	307
CHAPTER 17: Managing Quality and Risk	331
Part 5: Ensuring Success.....	355
CHAPTER 18: Building a Foundation	357
CHAPTER 19: De-Scaling across Teams.....	373
CHAPTER 20: Being a Change Agent	395
Part 6: The Part of Tens	421
CHAPTER 21: Ten Key Benefits of Agile Product Development.....	423
CHAPTER 22: Ten Key Factors for Agile Product Development Success	431
CHAPTER 23: Ten Signs That You're Not Agile	437
CHAPTER 24: Ten Valuable Resources for Agile Professionals	449
Index	455

Table of Contents

INTRODUCTION	1
About This Book.....	1
Foolish Assumptions.....	1
Icons Used in This Book	2
Beyond the Book.....	2
Where to Go from Here	3
PART 1: UNDERSTANDING AGILITY.....	5
CHAPTER 1: Modernizing Project Management	7
Project Management Needed a Makeover	8
The origins of modern project management	8
The problem with the status quo	9
Introducing Agile Project Management	11
How agile projects work.....	14
Agile Project Management Is Becoming Agile	16
Product Management.....	16
Differences between managing a project versus developing a product	16
Why agile product development works better	18
CHAPTER 2: Applying the Agile Manifesto and Principles	21
Understanding the Agile Manifesto	21
Outlining the Four Values of the Agile Manifesto.....	24
Value 1: Individuals and interactions over processes and tools.....	25
Value 2: Working software over comprehensive documentation	26
Value 3: Customer collaboration over contract negotiation.....	28
Value 4: Responding to change over following a plan.....	29
Defining the 12 Agile Principles.....	30
Agile principles of customer satisfaction.....	32
Agile principles of quality	34
Agile principles of teamwork.....	36
Agile principles of product development.....	38
Adding the Platinum Principles.....	42
Resisting formality.....	42
Thinking and acting as a team.....	43
Visualizing rather than writing.....	44
Changes as a Result of Agile Values	45
The Agile Litmus Test	47

CHAPTER 3: Why Being Agile Works Better	49
Evaluating Agile Benefits	49
How Agile Approaches Beat Historical Approaches	54
Greater flexibility and stability.....	55
Reduced nonproductive tasks.....	57
Higher quality, delivered faster.....	60
Improved team performance	61
Tighter control	62
Faster and less costly failure	63
Why People Like Being Agile	64
Executives	64
Product development and customers	65
Management	66
Development teams	67
CHAPTER 4: Agility Is about Being Customer Focused	69
Knowing Your Customers.....	69
Common methods for identifying your customer	71
Figuring Out the Problem Your Customer Needs to Solve	79
Using the scientific method	79
Failing early is a form of success	81
Defining customer-focused business goals.....	82
Story mapping	83
Liberating structures — simple rules to unleash a culture of innovation	83
Understanding Root Cause Analysis.....	84
Pareto rule	85
Five why's	86
Ishikawa (fishbone)	87
PART 2: BEING AGILE	89
CHAPTER 5: Agile Approaches	91
Diving under the Umbrella of Agile Approaches	91
Reviewing the Big Three: Lean, Scrum, and Extreme Programming	95
An overview of lean	95
An overview of scrum	100
An overview of extreme programming	105
Putting It All Together	107
CHAPTER 6: Agile Environments in Action.....	109
Creating the Physical Environment.....	110
Collocating the team	110
Setting up a dedicated area.....	112
Removing distractions	113

Low-Tech Communicating	114
High-Tech Communicating.....	116
Choosing Tools.....	118
The purpose of the tool	119
Tools that encourage the success of forced team dislocation	119
Organizational and compatibility constraints	121
CHAPTER 7: Agile Behaviors in Action.....	123
Establishing Agile Roles	123
Product owner	124
Development team member.....	128
Scrum master.....	130
Stakeholders.....	132
Agile mentor.....	134
Establishing New Values.....	134
Commitment	135
Focus	136
Openness	137
Respect	138
Courage.....	138
Changing Team Philosophy.....	139
Dedicated team	140
Cross-functionality.....	141
Self-organization	143
Self-management	144
Size-limited teams.....	146
Ownership	147
CHAPTER 8: The Permanent Team.....	149
Enabling Long-Lived Product Development Teams	149
Leveraging long-term knowledge and capability	150
Navigating Tuckman's phases to performance.....	151
Focusing on fundamentals.....	153
Creating a working agreement	154
Enabling Autonomy, Mastery, and Purpose	155
Autonomy.....	155
Mastery.....	155
Purpose.....	156
Highly aligned and highly autonomous teams	157
Building Team Knowledge and Capability	157

PART 3: AGILE PLANNING AND EXECUTION	159
CHAPTER 9: Defining the Product Vision and Product Roadmap	161
Agile Planning.....	162
Progressive elaboration	164
Inspect and adapt	165
Defining the Product Vision	165
Step 1: Developing the product objective	167
Step 2: Creating a draft vision statement.....	167
Step 3: Validating and revising the vision statement	169
Step 4: Finalizing the vision statement.....	170
Creating a Product Roadmap	171
Step 1: Identifying product stakeholders.....	172
Step 2: Establishing product requirements.....	173
Step 3: Arranging product features	175
Step 4: Estimating efforts and ordering requirements	176
Step 5: Determining high-level time frames	180
Saving your work.....	180
Completing the Product Backlog.....	180
CHAPTER 10: Planning Releases and Sprints	183
Refining Requirements and Estimates	183
What is a user story?.....	184
Steps to create a user story	186
Breaking down requirements	190
Estimation poker	192
Affinity estimating	195
Release Planning	197
Preparing for Release	200
Preparing the product for deployment	201
Prepare for operational support.....	201
Preparing the organization	203
Preparing the marketplace	204
Sprint Planning.....	205
The sprint backlog.....	206
The sprint planning meeting	207
CHAPTER 11: Working throughout the Day.....	215
Planning Your Day: The Daily Scrum	215
Tracking Progress	219
The sprint backlog.....	219
The task board.....	222

Agile Roles in the Sprint	224
Keys for daily product owner success	225
Keys for daily development team member success.....	226
Keys for daily scrum master success	227
Keys for daily stakeholder success	228
Keys for daily agile mentor success	228
Creating Shippable Functionality	229
Elaborating	230
Developing	230
Verifying	231
Identifying roadblocks	234
Information Radiators.....	235
The End of the Day	236
CHAPTER 12: Showcasing Work, Inspecting, and Adapting	239
The Sprint Review	239
Preparing to demonstrate	240
The sprint review meeting	241
Collecting feedback in the sprint review meeting.....	244
The Sprint Retrospective	245
Planning for retrospectives	247
The retrospective meeting.....	248
Inspecting and adapting.....	250
PART 4: AGILITY MANAGEMENT.....	251
CHAPTER 13: Managing a Portfolio: Pursuing Value over Requirements	253
Understanding the Differences in Agile Portfolio Management	254
Should we invest?	255
Factors for forecasting product investment returns	256
Managing Agile Product Portfolios	261
Should we continue investing?	266
Inspecting and adapting to the next opportunity.....	267
CHAPTER 14: Managing Scope and Procurement	269
What's Different about Agile Scope Management?	270
Managing Agile Scope.....	272
Understanding scope throughout product development.....	273
Introducing scope changes	275
Managing scope changes.....	275
Using agile artifacts for scope management.....	277
What's Different about Agile Procurement?	278
Managing Agile Procurement	280

Determining need and selecting a vendor.....	280
Understanding cost approaches and contracts for services.....	282
Working with a vendor	285
Closing a contract	286
CHAPTER 15: Managing Time and Cost	287
What's Different about Agile Time Management?	287
Managing Agile Schedules	289
Introducing velocity.....	289
Monitoring and adjusting velocity.....	291
Managing scope changes from a time perspective	297
Managing time by using multiple teams	298
Using agile artifacts for time management.....	298
What's Different about Agile Cost Management?.....	299
Managing Agile Budgets.....	300
Creating an initial budget.....	301
Creating a self-funding product.....	302
Using velocity to determine long-range costs.....	303
Using agile artifacts for cost management	306
CHAPTER 16: Managing Team Dynamics and Communication.....	307
What's Different about Agile Team Dynamics?	307
Managing Team Dynamics.....	309
Becoming self-managing and self-organizing	310
Supporting the team: The servant-leader	314
Working with a dedicated team.....	316
Working with a cross-functional team	317
Reinforcing openness	319
Limiting development team size.....	320
Managing product development with dislocated teams.....	321
What's Different about Agile Communication?.....	324
Managing Agile Communication	325
Understanding agile communication methods	325
Status and progress reporting.....	328
CHAPTER 17: Managing Quality and Risk	331
What's Different about Agile Quality?	331
Managing Agile Quality.....	334
Quality and the sprint	335
Proactive quality	335
Quality through regular inspecting and adapting.....	341
Automated testing.....	342

What's Different about Agile Risk Management?	345
Managing Agile Risk	348
Reducing risk inherently.....	348
Identifying, prioritizing, and responding to risks early.....	353
PART 5: ENSURING SUCCESS	355
CHAPTER 18: Building a Foundation	357
Organizational and Individual Commitment.....	357
Organizational commitment	358
Individual commitment	359
Getting commitment.....	360
Can you make the transition?	361
Timing the transition.....	362
Choosing the Right Pilot Team Members	363
The agile champion.....	363
The agile transition team	364
The product owner	365
The development team	366
The scrum master	366
The stakeholders.....	367
The agile mentor	367
Creating an Environment That Enables Agility	368
Support Agility Initially and Over Time.....	371
CHAPTER 19: De-Scaling across Teams	373
Multi-Team Agile Development.....	374
Making Work Digestible through Vertical Slicing	376
Scrum of scrums	376
Multi-Team Coordination with LeSS	380
LeSS, the smaller framework.....	380
LeSS Huge framework.....	381
Sprint review bazaar	382
Observers at the daily scrum.....	383
Component communities and mentors.....	383
Multi-team meetings.....	383
Travelers	384
Aligning through Roles with Scrum@Scale	384
The scrum master cycle	385
The product owner cycle	387
Synchronizing in one hour a day.....	388
Joint Program Planning with SAFe.....	388
Joint program increment planning	391
Clarity for managers	392
Disciplined Agile Toolkit	392

CHAPTER 20: Being a Change Agent	395
Becoming Agile Requires Change	395
Why Change Doesn't Happen on Its Own	396
Strategic Approaches to Implementing and Managing Change	397
Lewin	398
ADKAR's five steps to change	399
Kotter's eight steps for leading change	400
Platinum Edge's Change Roadmap	401
Step 1: Conduct an agile audit to define an implementation strategy with success metrics	403
Step 2: Build awareness and excitement	404
Step 3: Form a transformation team and identify a pilot	405
Step 4: Build an environment for success	407
Step 5: Train sufficiently and recruit as needed	408
Step 6: Kick off the pilot with active coaching	408
Step 7: Execute the Roadmap to Value	410
Step 8: Gather feedback and improve	410
Step 9: Mature and solidify improvements	411
Step 10: Progressively expand within the organization	412
Leading by Example	412
The role of a servant-leader in an agile organization	413
Keys for successful servant leadership	413
Avoiding Transformation Pitfalls	414
Avoiding agile leadership pitfalls	417
Signs Your Changes Are Slipping	418
PART 6: THE PART OF TENS	421
CHAPTER 21: Ten Key Benefits of Agile Product Development	423
Higher Customer Satisfaction	423
Better Product Quality	424
Reduced Risk	425
Increased Collaboration and Ownership	426
More Relevant Metrics	426
Improved Performance Visibility	427
Increased Investment Control	428
Improved Predictability	429
Optimized Team Structures	429
Higher Team Morale	430

CHAPTER 22: Ten Key Factors for Agile Product Development Success.....	431
Dedicated Team Members.....	431
Collocation	432
Done Means Shippable.....	433
Address What Scrum Exposes.....	433
Clear Product Vision and Roadmap	433
Product Owner Empowerment.....	434
Developer Versatility.....	434
Scrum Master Clout	435
Leadership Support for Learning	435
Transition Support	436
CHAPTER 23: Ten Signs That You're Not Agile.....	437
A Non-Shippable Sprint Product Increment	437
Long Release Cycles	438
Disengaged Stakeholders.....	439
Lack of Customer Contact	440
Lack of Skill Versatility.....	441
Automatable Processes Remain Manual.....	442
Prioritizing Tools over the Work	442
High Manager-to-Creator Ratio.....	444
Working around What Scrum Exposes.....	445
Practicing Faux Agile	446
CHAPTER 24: Ten Valuable Resources for Agile Professionals	449
Agile Project Management For Dummies Online Cheat Sheet.....	449
Scrum For Dummies.....	450
The Scrum Alliance	450
The Agile Alliance.....	450
International Consortium for Agile (ICAgile)	451
Mind the Product and ProductTank	451
Lean Enterprise Institute	451
Extreme Programming	452
The Project Management Institute Agile Community	452
Platinum Edge	452
INDEX.....	455

Introduction

Welcome to *Agile Project Management For Dummies*, 3rd Edition. Agile project management has grown to be as common as any management technique for product development — and not only software product development. For nearly two decades, we have trained and coached companies big and small, all over the world, about how to become more nimble, adaptive, and responsive in both the development of their products and their organizations — in other words, how to become more agile. Through this work, we found there was a need to write a digestible guide that anyone, regardless of experience, could understand.

About This Book

Agile Project Management For Dummies, 3rd Edition is more than just an introduction to agile practices and approaches; you also discover the steps to become more agile in mindset and behavior. The material here goes beyond theory and is meant to be a field guide for all experience levels, giving you the tools and information you need to be successful with agile techniques in the trenches of product development.

Foolish Assumptions

This book was written as a reference guide for anyone wanting to learn more about business agility. If you strive to be more agile in responding to customer needs and problems — whether or not you're an organizational leader, a project manager, a member of a product team, an agile enthusiast, or a product stakeholder — this book will help you on your journey.

Regardless of your experience or level of familiarity, this book provides insights you may find helpful. We hope it brings clarity to any confusion or myths regarding agile product development 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 product development 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. 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 product development, 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/agileprojectmanagementfd3e.

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 products under an agile approach?" check out Chapter 24 or head over to www.dummies.com for more resources.

We've provided a cheat sheet for tips on assessing your current product development efforts in relation to agile principles as well as free tools for managing projects using agile techniques. To get to the cheat sheet, go to 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 product development and agile approaches, start with Chapter 1 and read the book straight through to the end.
- » If you're a member of a product team and want to know the basics of agile product development, check out the information in Part 3 (Chapters 9 through 12).
- » If you're a project manager transitioning to agile approaches to product development, you may be interested to learn how agile techniques improve the management of time, cost, scope, procurement, quality, and risk. Review Part 4 (Chapters 13 through 17).
- » If you know the basics of agile product development and are looking at bringing agile practices to your company or expanding your agile footprint across your organization, Part 5 (Chapters 18 through 20) will provide you with helpful information.

1 **Understanding Agility**

IN THIS PART . . .

Understand why project management has modernized due to the flaws and weaknesses in historical approaches to project management.

Find out why agile methods are becoming more product-focused than project-focused, and become acquainted with the foundation of agile product development: the Agile Manifesto and the 12 Agile Principles.

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

Understand the importance of placing the customer's needs first and why agile techniques help to make the customer central to every decision, functionality, and problem.

IN THIS CHAPTER

- » Understanding why project management needs to change
- » Seeing how agile project management is becoming agile product management
- » Finding out about agile product development

Chapter **1**

Modernizing Project Management

Agile is a descriptor of a mindset approach to project management that focuses on early delivery of business value, continuous improvement of the product being created and the processes used to create the product, 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 products and services, and executives whose companies fund product development. While business agility is popular in software product development, agile values, principles, and techniques apply in a multitude of industries and applications — not just software. This chapter also explains the advantages of agile approaches over long-standing project management methodologies.

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're reading this book, you're likely 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 product development, 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 been around only 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.

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

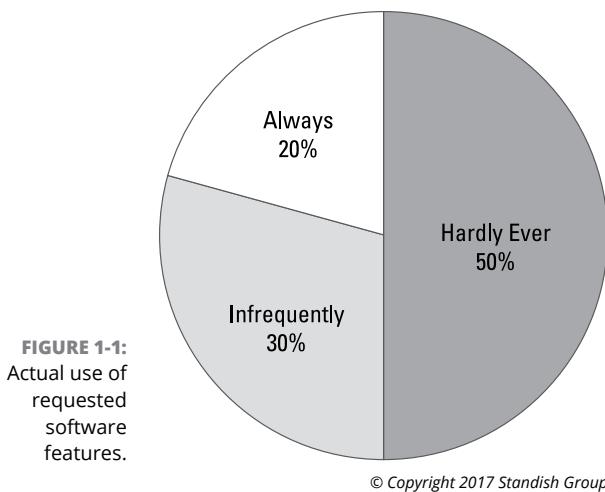
Computer technology has, of course, changed a great deal since the last century. Many people have a computer on their wrist with more power, memory, and capabilities than the largest, most expensive machine that existed when people first started using waterfall methodologies.

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 tablet or smartphone, 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 — 2-year-olds are almost more adept with the iPhone than their parents. The demand for newer, better 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. 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 80 percent of requested features are infrequently or never used.



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.

SOFTWARE PROJECT SUCCESS AND FAILURE

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

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 three decades, people working on projects have recognized the growing problems with traditional project management and have been working to create a better model.

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.

Agile History

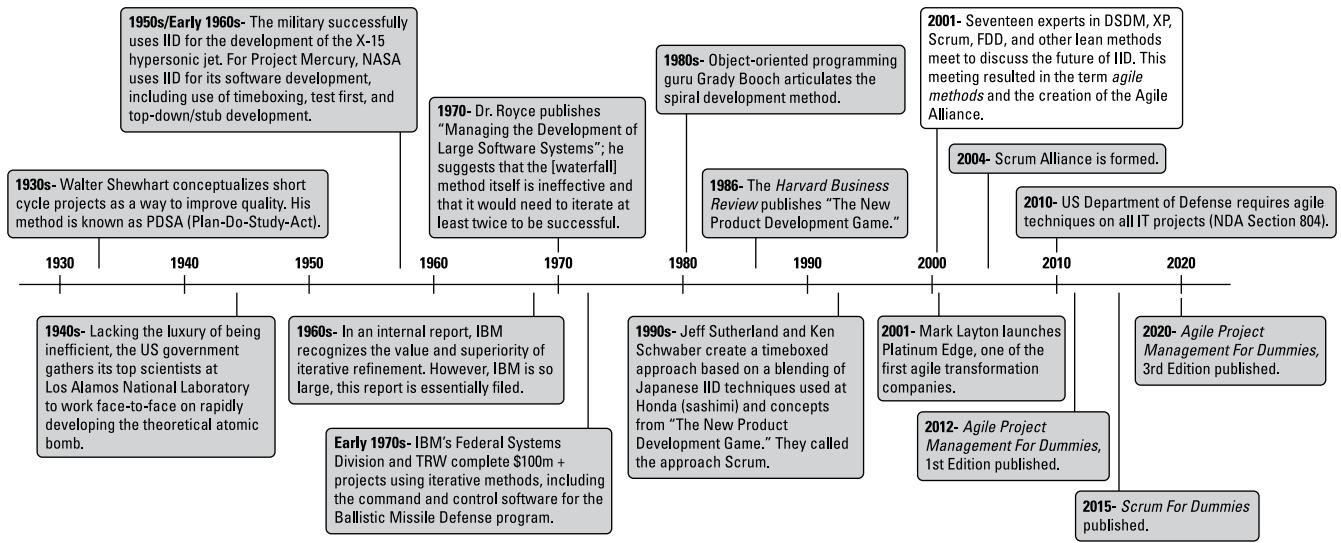


FIGURE 1-2: Agile project management timeline.

In 1986, Hirotaka Takeuchi and Ikujiro Nonaka published an article called “The 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* referred to a player formation in rugby.) Scrum eventually became one of the most popular agile frameworks for delivering value to customers.

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 *Manifesto for Agile Software Development* (commonly referred to as 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 12 principles behind the Agile Manifesto 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 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.



Martin Fowler, one of the co-authors of the Agile Manifesto, writes that many different words were discussed for naming their movement. They considered *lightweight methods*, *adaptive*, and many others until landing on *agile* as the best descriptor of the adaptiveness and responsiveness to change they were seeking.

Other synonyms are resilient, nimble, and healthy. When you think of agile, think healthy. Healthy organizations and teams are agile, resilient, nimble, and responsive.

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 product development effort involves the same type of work as in a traditional waterfall project: You create requirements and designs, develop product features, document what was done and why, and continuously integrate new features. You test the product, fix any problems, and deploy the product 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.



WARNING

Mixing traditional project management methods with agile approaches is like saying, “I have a Tesla Model S. However, I’m using a wagon wheel on the front left side. How can I make my car as fast and efficient as the other Teslas?” 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.

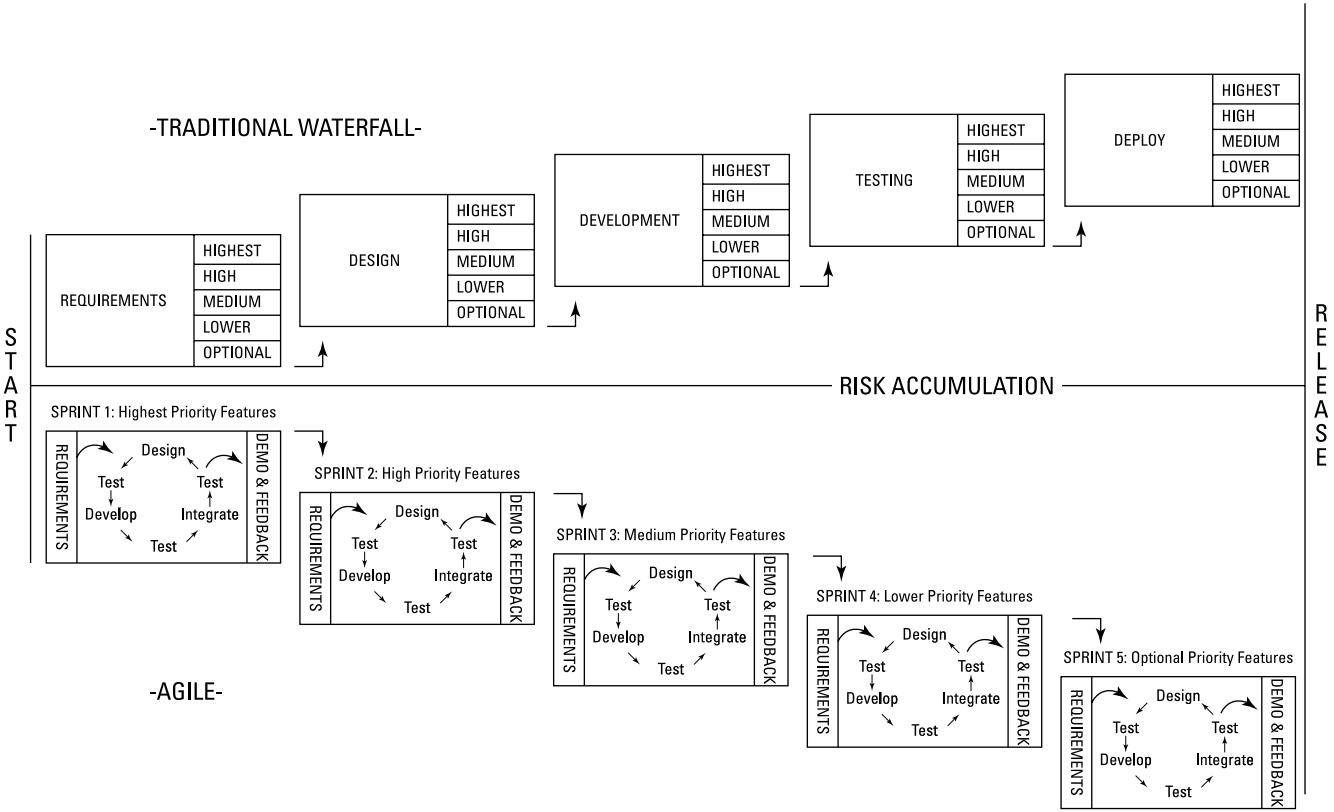


FIGURE 1-3: Waterfall versus agile project.

Agile Project Management Is Becoming Agile Product Management

Traditional projects, by definition, are organized into temporary, build-only team efforts designed to accomplish specific benefits projected in a business case. Project initiation, which is when we know the least, is when budgets, schedules, and expectations are set. When projects end, the teams are disbanded and unfamiliar operational teams are left to support the customer and product. If more work is required, new team members are allocated to a new project and must refamiliarize themselves with the product architecture. Projects typically end once deliverables are released into production, leaving others to support and evaluate the effect on business.

Today, products are considered long-term, value-creating assets requiring permanent teams who iteratively elaborate, design, develop, test, integrate, document, and even support products until business outcomes are achieved. A high-performing team continuously inspects and adapts until the customer's problems are solved, and the team retains this hard-earned knowledge. The team and customer collaborate to create value over following written specifications.

More and more we see a project-driven approach as a constraint to delivering customer value early and often. Taking an agile approach to product development limits you not to time or money but to value. Organizations most effectively deliver value when they use the following formula to determine when it's time to shift priorities:

$$AC + OC > V, \text{ that is, actual cost} + \text{opportunity cost} > \text{value}$$

When the actual cost of working on a product's additional requirements plus the opportunity cost of not working on a different investment opportunity exceed the expected value of delivering those remaining product requirements, the team shifts to developing the more valuable investment opportunity.

Differences between managing a project versus developing a product

Three primary differences exist between managing a project and developing a product:

- » Products benefit most from stable, long-lived, and even permanent teams.
- » Products can be not only short-term assets but also long-term assets. Active products are never really finished because they require maintenance and improvements.
- » Products are part of a portfolio designed to maximize value over following specifications.

Permanent team over temporary team

Long-lived products are best developed and maintained by long-lived and even permanent teams. The longer a team works together iteratively building an emergent architecture and expanding its capability and high performance, the better the team understands the customer and the more predictable the team becomes. Project-focused teams come together for a specific amount of time and then move on to something new. Lessons learned at the end of a project may not even apply to the next project because the context of people, technologies, and customers will most likely be different. Stable permanent teams enable transparency, inspection, and adaptation (empirical process control).



TIP

Permanent doesn't mean that agile product teams don't change and career aspirations are inhibited. However, team personnel changes are an exception rather than the rule. People — especially people who become more valuable because of their expanding capability — are presented with career-enhancing opportunities. Ideally, permanent team members behave more like a family than as a temporary, one-project-only group.

Products as long-term assets rather than project deliverables

Product development is risky. Uncertainty abounds at every corner. But uncertainty is what makes agile product development ideal! Traditional projects are tasked with accomplishing specific system deliverables within a fixed time frame, but agile product development iteratively reduces uncertainty by building useable, fully functional product increments, gathering and implementing feedback throughout development. The product becomes a customer-aligned, problem-solving asset. Active products are never finished because maintenance must be performed and improvements can be made.

Investments in time, money, and people, particularly with today's capital expenditure strategies, change products into depreciable assets that improve bottom-line results for not only revenue but also cost savings. Treating product

development as an asset creator rather than a cost expenditure changes the perspective of everyone involved. Continuous delivery of customer value through agile product development increases the likelihood of additional funding.



TECHNICAL
STUFF

Capital expenditures, commonly known as *CapEx*, are funds used by a company to acquire, upgrade, and maintain physical assets such as property, buildings, industrial plants, technology, and equipment. *CapEx* is often used to undertake new projects or investments by the firm.

Pursuing value over specifications

Early failure is a key tenant of agility. Agile teams are obsessed with taking risks to create customer value. Like scientists, they create a hypothesis, test it in the real world, evaluate the results, and then adjust the hypothesis and test it again. They repeat this process again and again, aligning the product more closely to the customer's needs with each iteration. Teams trade reams of documented specifications for real-world feedback from the customer. With agile product development, functionality priorities are set by the people most familiar with the problem to be solved.

Why agile product development works better

Throughout this book, you see how product development with an agile approach works better than with a traditional approach. Agile approaches can produce more successful products. 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 with agile techniques. The decrease in failure for agile product development is a result of agile teams making immediate adaptations based on frequent inspections of progress and customer satisfaction.

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

- » **Project success rates:** In Chapter 17, you find out how the risk of catastrophic project failure falls to almost nothing with agile product development. Agile approaches of prioritizing by business value and risk ensure early success or failure. Agile approaches to testing throughout the product development help ensure that you find problems early, not after spending a large amount of time and money.

- » **Scope creep:** In Chapters 9, 10, and 14, you see how agile approaches accommodate changes throughout product development, minimizing scope creep. Following agile principles, 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.
- » **Inspection and adaptation:** In Chapters 12 and 16, you find details of how regular inspections and adaptation work throughout agile product development. Agile 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 business agility can help you gain control of product outcomes. 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.

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

Chapter 2

Applying the Agile Manifesto and Principles

This chapter describes the basics of what it means to be agile as outlined in 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 helping organizations improve their business agility.

This foundation provides product development teams with the information needed to evaluate whether the team is following agile principles, as well as whether its 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. To learn more about development teams, see Chapter 7.

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 product development in every industry, not just software.

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

- » **The Agile Manifesto (originally the Manifesto for Agile Software Development):** An intentionally streamlined expression of core development values
- » **The Agile Principles:** A set of 12 guiding concepts that support product development teams in delivering value and staying on track
- » **The Agile Alliance:** A community development organization focused on supporting individuals and organizations 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 hierarchical-based control, the manifesto focuses on

- » People
- » Communications
- » Product
- » Flexibility

The Agile Manifesto represents a big shift in focus in how products 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 to 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 80 percent of features developed under a waterfall model are infrequently or never used (as discussed in Chapter 1). Starting with a plan is vital, but the start is when we know the least. Product development teams don't plan less than waterfall teams — they plan as much or more. However, teams take a just-in-time approach, planning just enough when needed in support of a strategic product vision and roadmap. Adaptation of the plan to the realities along the way is how teams avoid wasteful functionality and 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 techniques have spread beyond software development and even outside computer-related products. Today, agile approaches such as scrum are disrupting biotech, manufacturing, aerospace, engineering, marketing, building construction, finance, shipping, automotive, utility, and energy industries with companies such as Apple, Microsoft, and Amazon leading the way. If you want early empirical feedback on the product or service you're providing, you can benefit from agile methods.

The *State of Scrum 2017–2018* report quoted a Scrum Alliance board member who said, “Any organization that does not go through an Agile transformation will die. It is the same as a company refusing to use computers.”



REMEMBER

The Agile Manifesto and 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. Agile values and principles apply to all product development activities, not just software.

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?

Although the agile values and principles are not numbered, we've numbered them here and throughout the book for ease of reference. The numbering matches their order in the manifesto.

Value 1: Individuals and interactions over processes and tools

When you allow each person to contribute his or her unique value to a product, 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 product 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 interactions and valuing processes and tools.



You can find a blank template of Table 2-1 on the book's companion website at <http://www.dummies.com/go/agileprojectmanagementfd3e>. Jot down the pros and cons of each approach that apply to you and your products.



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 product 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.

TABLE 2-1

Individuals and Interactions versus Processes and Tools

	Individuals and Interactions Have High Value	Processes and Tools Have High Value
Pros	<p>Communication is clear and effective.</p> <p>Communication is quick and efficient.</p> <p>Teamwork becomes strong as people work together.</p> <p>Development teams can self-organize.</p> <p>Development teams have more chances to innovate.</p> <p>Development teams can quickly adjust processes as necessary.</p> <p>Development team members can take personal ownership of the product.</p> <p>Development team members can have deeper job satisfaction.</p>	<p>Processes are clear and can be easy to follow.</p> <p>Written records of communication exist.</p>
Cons	<p>To enable more team empowerment and less command and control, managers may have to unlearn traditional leadership tendencies.</p> <p>People may need to let go of ego to work well as members of a team.</p>	<p>People may over-rely on processes instead of finding the best ways to create good products.</p> <p>One process doesn't fit all teams — different people have different work styles.</p> <p>One process doesn't fit all products.</p> <p>Communication can be ambiguous and time-consuming.</p>

Value 2: Working software over comprehensive documentation

A development team's focus should be on producing working functionality. With agile development, 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 investment.

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 wouldn’t have any working software to give the customer; 75 percent done traditionally means you are 75 percent in progress and 0 percent done. With

agile product development, however, by using the definition of done, you would have working, potentially shippable functionality for 75 percent of your product 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 whether documents produced on a recent project you were involved in added value to the functionality being delivered to your customer.



REMEMBER

With agile product development, *barely sufficient* is a positive description, meaning that a task, document, meeting, or almost anything created 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*, or adding unnecessary frivolity — and effort — to a feature, task, document, meeting, or anything else.

All development requires some documentation. With agile product development, documents are useful only if they support 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/agileprojectmanagementfd3e. Use that form to assess how well your documentation directly contributed to the product and whether it was barely sufficient.



TIP

We'll often stop producing a document and see who complains. After we know the requester of the document, we'll 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.

Product development 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 teams to understand requirements and assess real-time status daily. With agile approaches, teams spend more time on development and less time on documentation, resulting in a more efficient delivery of a working product. See Chapter 9 for more on the product backlog, and see Chapter 10 to learn more about the sprint backlog.

TABLE 2-2**Identifying Useful Documentation**

Document	Does the Document Add to Product Value?	Is the Document Barely Sufficient or Gold-Plated?
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 may spend a lot of time creating and updating project schedules, team members tend to want to know only key deliverable dates. Management often wants to know only whether the deliverable is on time, ahead of schedule, or behind.
Requirements documentation	Yes. All products 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 enable product requirement conversations.
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 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.

Value 3: Customer collaboration over contract negotiation

The customer is not the enemy. Really.

Historical project management approaches usually limit customer involvement to a few development stages:

- » **Start of a project:** When the customer and the project team negotiate contract details.
- » **Any time the scope changes during the project:** When the customer and the project team 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 team and the customer negotiate additional changes to the contract.

This historical focus on negotiation, avoidance of scope change, and limitation of direct customer involvement 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 its start. Locking product details into a contract at the beginning of development means you have to make decisions based on incomplete knowledge. If you have flexibility for change as you learn more about a product and the customer the product is serving, 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 product development on an ongoing basis.

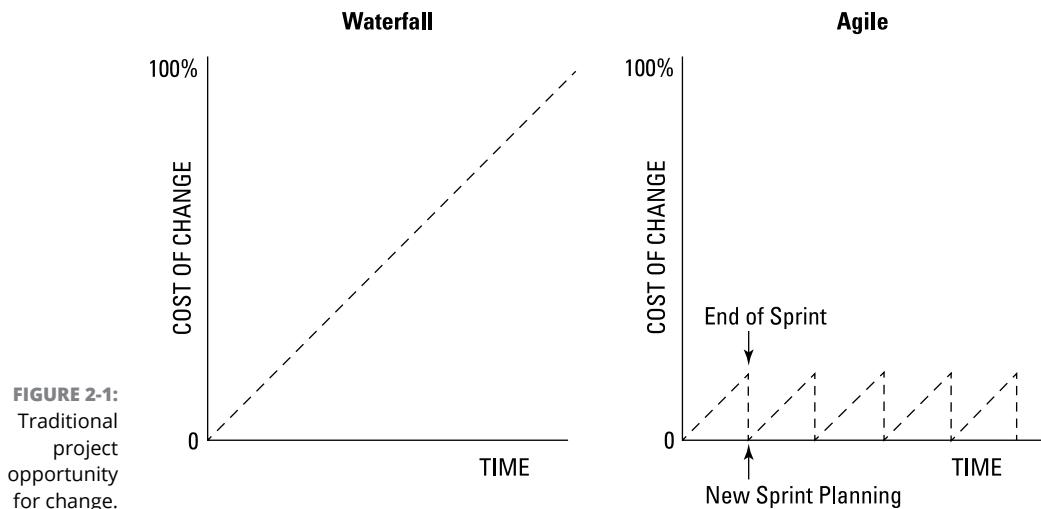
Using an agile approach in practice, you'll experience a partnership between the customer and the product development team in which discovery, questioning, learning, and adjusting during the course of development are routine, acceptable, and systematic. This partnership results in superior products better suited for the customer's needs.

Value 4: Responding to change over following a plan

Change is a valuable tool for creating great products. 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 or, even worse, unable to react timely to changing market conditions.

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 decrease, and costs more.



By contrast, agile development accommodates change systematically. The flexibility of agile approaches increases stability because product changes are predictable and manageable — in other words, it's expected and nondisruptive to a product development team. In later chapters, you discover how agile approaches to planning, working, and prioritization allow teams to respond quickly to change.

As new events unfold, the team incorporates these 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 the transition to agile approaches, they augmented the four values of the Agile Manifesto with 12 principles.



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 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
- » Product development

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 of a product in a number of ways:

- » The customer is the person or group paying for the product.
- » In some organizations, the customer may be a client, external to the organization.
- » In other organizations, the customer may be a stakeholder or a group of 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, we call that person the *user*.

How do you enable these principles? Consider the following:

- » Scrum teams (*scrum* is a popular agile framework you learn more about in Chapter 5) include a *product owner*, a person who is responsible for translating what the customer wants into product requirements. To learn more about the role of the product owner, see Chapter 7.
- » 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 product development.
- » Frequent delivery of working product features 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 one 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 development, rather than waiting until the end for the first, and maybe only, delivery of releasable product features.

In Table 2–3, we list some customer satisfaction issues that commonly arise during product development. Use Table 2–3 and gather some examples of customer dissatisfaction that you’ve encountered. Do you think becoming more agile would make a difference? Why or why not?

TABLE 2-3 Customer Dissatisfaction and How Agile Might Help

Examples of Customer Dissatisfaction with Product Development	How Agile Approaches Can Increase Customer Satisfaction
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>Product development 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 can provide feedback at the end of the sprint, not at the end of development, when the feedback would be too late.</p>
The product wasn't delivered when the customer needed it.	Working in sprints allows 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/agileprojectmanagementfd3e.



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 team in the same place to eliminate communication barriers
- » Breaking requirements into small chunks of value that can be delivered in short iterations
- » Keeping written requirements simple, forcing more robust and effective face-to-face communication
- » Getting the product owner's acceptance as soon 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

A product development team commits to producing quality in every product increment it creates — from development through documentation to integration and test results — every day. Each 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 of technical quality and be empowered to solve problems. They carry the responsibility for determining how to create the product, deciding the technical work needed to create it, and organizing product development. People not doing the work don't tell them how to do it.
- » With software development, an agile approach requires architectures that make coding and testing 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 because everything works on paper. 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 the development of the requirement was completed. Feedback from the product owner is often necessary even during the development of a feature.
- » At the end of every iteration (lasting two weeks or less for most teams), working functionality 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. As much as possible, testing is automated. To learn more about automated testing, see Chapter 17.
- » With software development, ensuring new code is tested and integrates with previous versions occurs in small increments, possibly 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.
- » With software development, examples of technical excellence include establishing coding standards, using service-oriented architecture, implementing automated testing, and building for future change.



REMEMBER

Agile principles apply to more than software products. Technical excellence is crucial whether you're developing marketing campaigns, publishing books, involved in manufacturing, or engaged in research and development. All disciplines have a set of technical practices that teams can use to build in quality all along the way.



TIP

Agile approaches provide the following strategies for quality management:

- » Defining what *done* means (that is, shippable) at the beginning of development and then using that definition as a benchmark for quality
- » Testing aggressively and daily through automated means
- » Building only the functionality 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 product lifecycle

Agile principles of teamwork

Teamwork is critical to agile product development. Creating good products requires cooperation among all members of the team, including customers and stakeholders. Agile approaches support team-building and teamwork, and they emphasize trust in self-managing development teams. A permanent, skilled, motivated, unified, and empowered team is a successful team. To learn more about permanent teams, see Chapter 8.

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 product development. 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 each team member's mind sharp and product quality high. This fact was known as early as 1908, when Dr. Ernst Abbe quantified that reducing daily work from 12 to 8 hours increased cumulative output. P. Sargent Florence, author of *The Economics of Fatigue and Unrest*, showed that 8-hour days produce between 16 to 20 percent higher total output than 9-hour days.

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 team members responsible as a single team, not as individuals.
- » Use face-to-face communication to quickly and efficiently convey information.



WARNING

Suppose that you usually communicate to Sharon by email. 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 another 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. A five-minute discussion addresses the issue quickly and with less risk of misunderstanding — and with a reduced cost of delay.

- » 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. Teams who rely on written communication for collaboration are slower and more prone to miscommunication errors. Written intra-team communication is a liability.
- » Make sure that *lessons learned* is an ongoing feedback loop rather than an end-of-project-only occurrence. Retrospectives should be held at the end of each iteration, when reflection and adaptation can improve development team productivity immediately going forward, creating ever higher levels of efficiency. A lessons learned meeting at the end of development has minimal value because the next product created might have a different group of people and practices. To learn more about retrospectives, see Chapter 12.



TIP

The first retrospective is as valuable (or even more valuable) as any future retrospective because, at the beginning, the team has the opportunity to make changes that can benefit the rest of the product development moving forward.

The following strategies promote effective teamwork:

- » Collocate the development team so it has no physical barriers to effective and real-time communication.
- » Put together a physical environment 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 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 team to solve problems rather than having managers solve problems for the team.
- » Resist the temptation to shuffle team members. Allow the team to become a stable, permanent, high-performing, capability-expanding team.



REMEMBER

A long-term product perspective requires long-term, permanent teams. High-performing teams take years to build. Their understanding of the customer, their feedback from each release, the product support they provide, and the product development environment logically encourage teams to remain as stable as possible. Team members may seek new opportunities for career development outside the team, but for the most part teams should remain as constant as possible for maximum value. As each new feature is built, the team remains constant, able to support and learn from the product's adoption by the customer.

Agile principles of product development

Agility in product 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 product'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 product development is clear to everyone.

All 12 principles support product management, but principles 1–3 and 7–10 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.
- 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.

Following are some advantages of adopting agile product management:

- » Product development teams achieve faster time-to-market and, consequently, cost savings. They start development earlier than in traditional approaches because agile approaches minimize the exhaustive up-front planning and documentation that is conventionally part of the early stages of a waterfall project.
- » Product 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 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 goal we can set that still adds value?” instead of focusing on including all 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 minimizing complexity of what’s under the hood (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 (that is, sufficient but just barely), and you’ll have more time to focus on supporting the development team.

- » By encapsulating development into short sprints that last several 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 development to provide a predictable rhythm 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 software development set of practices called extreme programming (XP), the maximum workweek is 40 hours, and the preferred workweek is 35 hours. Agile product development is constant and sustainable, as well as more productive, especially long term.



WARNING

Traditional approaches routinely feature a *death march*, in which the team puts in extremely long hours for days and even weeks at the end 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 with the existing product, 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 traditional project before, you might have a basic understanding of project management activities. In Table 2-4, we list a few 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 look different from traditional project management.



ON THE
WEB

A blank template of Table 2-4 is available at www.dummies.com/go/agileprojectmanagementfd3e.

TABLE 2-4

Contrasting Historical Project Management with Agile Product Management

Traditional Project Management Tasks	Agile Approach to Product Development Tasks
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 product development.
Conduct weekly status meetings with all project stakeholders and developers. Send 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. The team can update the centrally visible burndown chart in under a minute at the end of each day. Anyone, including stakeholders, can see the real-time progress on demand.
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 removing impediments and distractions. Development teams define and pull (as opposed to push) their own tasks.



TIP

Successful product development is facilitated by the following agile approaches:

- » Supporting the development team with real-time answers to its questions, shielding it from competing priorities, and empowering it to develop solutions and determine how much work to take on in each iteration
- » 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 team from organizational disruptions that could undermine product goals by introducing work that is not relevant to the product objectives
- » Understanding that an appropriate balance between work and life is a component of efficient development
- » Viewing the product as a long-term investment requiring permanent teams pursuing value over specifications

Adding the Platinum Principles

Through in-the-trenches experience working with teams transitioning to agile product development — and field testing in large, medium, and small organizations worldwide — we developed three additional principles of agile product development that we call the Platinum principles:

- » 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 product development teams can drift toward excessive formalization. For example, it isn't uncommon for us to find 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 demobilization and remobilization 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 product, not hamper it; and flashy displays, such as well-decorated presentations, are avoided. Professional, frank communications are best for the team, and the entire organizational environment has to make that openness available and comfortable.



TIP

Strategies for success in resisting formality include the following:

- » Reducing organizational hierarchy wherever possible by eliminating titles in the team
- » Avoiding aesthetic investments such as elaborate slide presentations or extensive meeting minute forms, especially when demonstrating shippable functionality at the end of a sprint
- » Educating stakeholders who request complicated displays about the high costs and low returns of such displays

Thinking and acting as a team

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 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 (each partner is knowledgeable in the area) and shadowing (only one partner is knowledgeable in the area) raise product quality and team member capability and reduce single points of failure. You can learn more about pair programming in Chapter 17.

- » Replace individual work titles with a uniform product developer title. Development activities include all tasks necessary to take requirements through to functionality, including design, implementation (for example, coding), testing, and documentation — not just writing code or turning a screwdriver.
- » Report at the team level only, as opposed to creating special management reports that subdivide the team.
- » Replace individual performance metrics with team performance metrics.

Visualizing rather than writing

A product development 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 the description is 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 status through charts, graphs, and dashboards, such as those in Figure 2-2

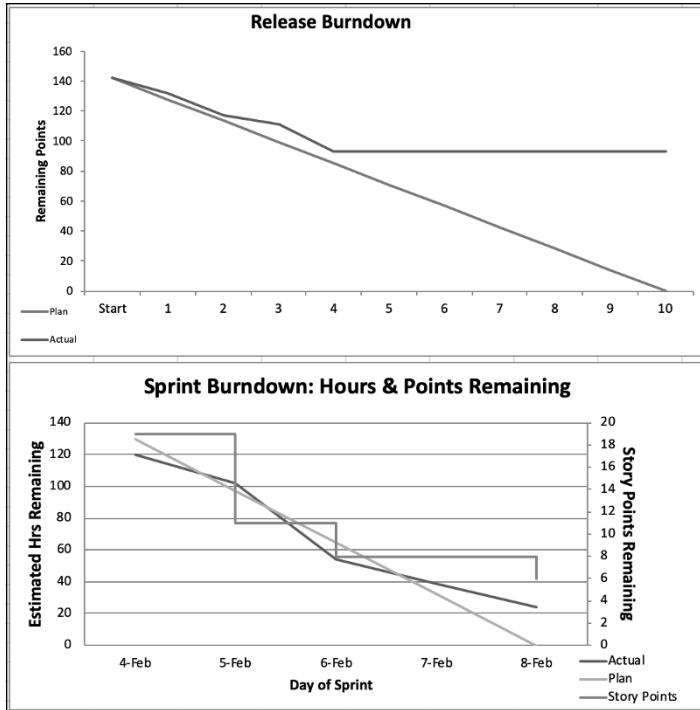


FIGURE 2-2:
Charts and graphs for providing transparency.

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 product 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 product. The same product created by different team members will be a different product.

- » **Agile approaches changed the relationship between business and IT groups.** Agile product development addressed the problems associated with the historical separation between business and IT by bringing these contributors together on the same 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 the last century's traditions and develop a structure at all levels that continually asks, "What's best for the customer, the product, and the team?"

A product development 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 products, customer-driven solutions, and entire organizations more productive and profitable. This evolution will be driven by passionate methodologists 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 change 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 team?
- 5.** Does our environment give the team the support it needs to get the job done?
- 6.** Are we communicating in person face-to-face rather than over the phone or through email?
- 7.** Are we measuring progress by the amount of working functionality produced?
- 8.** Can we maintain our current pace of development 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 product goal for our customer?
- 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 likely becoming more agile. 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 for yourself, as well as with your team and the wider organization.

IN THIS CHAPTER

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

Chapter 3

Why Being Agile Works Better

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 product development advantages, the bottom line is twofold: success and stakeholder satisfaction.

Evaluating Agile Benefits

The agile concept of product development is different from previous project management approaches and 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 principles provide a framework for how we *want* to work — how we naturally function when we solve complex problems.

Historical methods of project management were developed not for contemporary development lifecycles, such as new product development, but for simpler and obvious systems. It's no wonder that these project management methods don't fit

when attempting to build more complex, modern products, such as artificial intelligence, aircraft, cybersecurity, medical devices, financial management systems, mobile applications, and 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 development. 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.



You can use agile product development 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 <https://tomwujec.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 up-front 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 developing a product) 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.

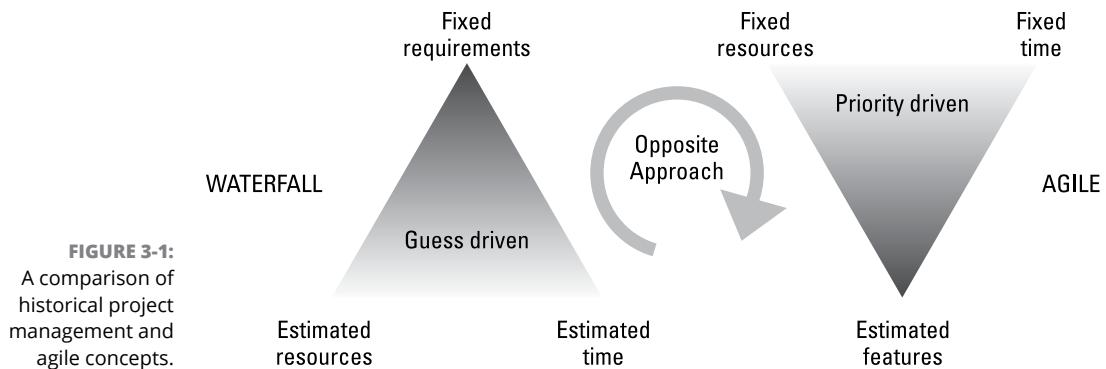


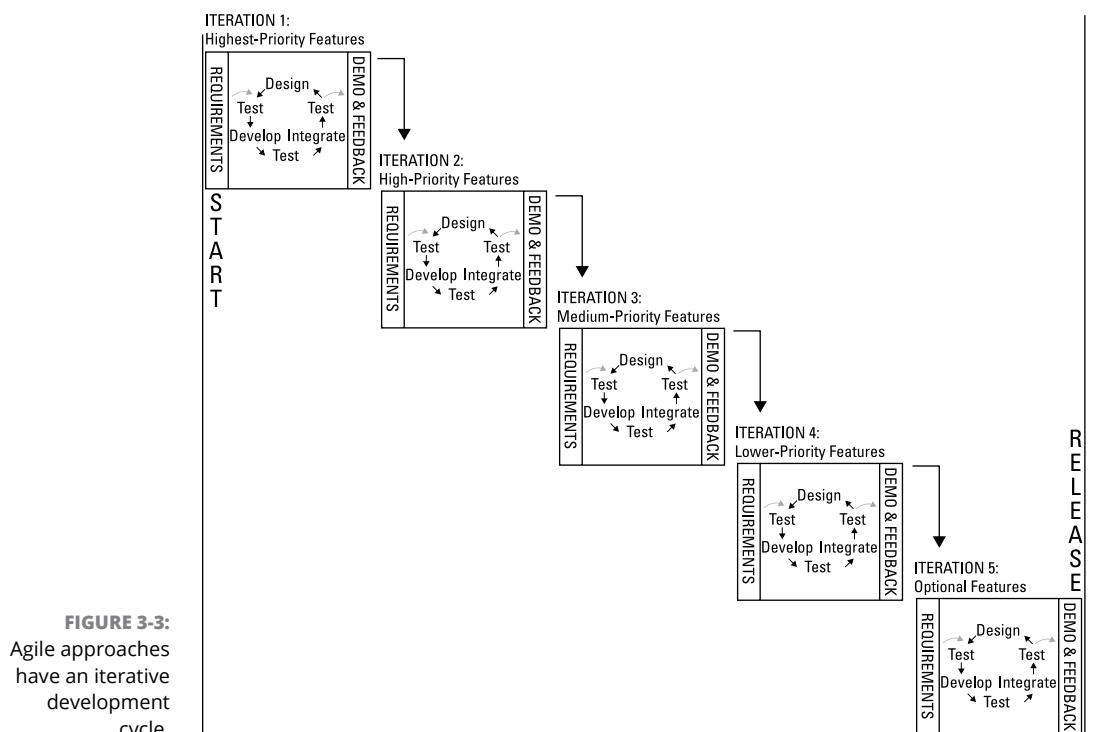
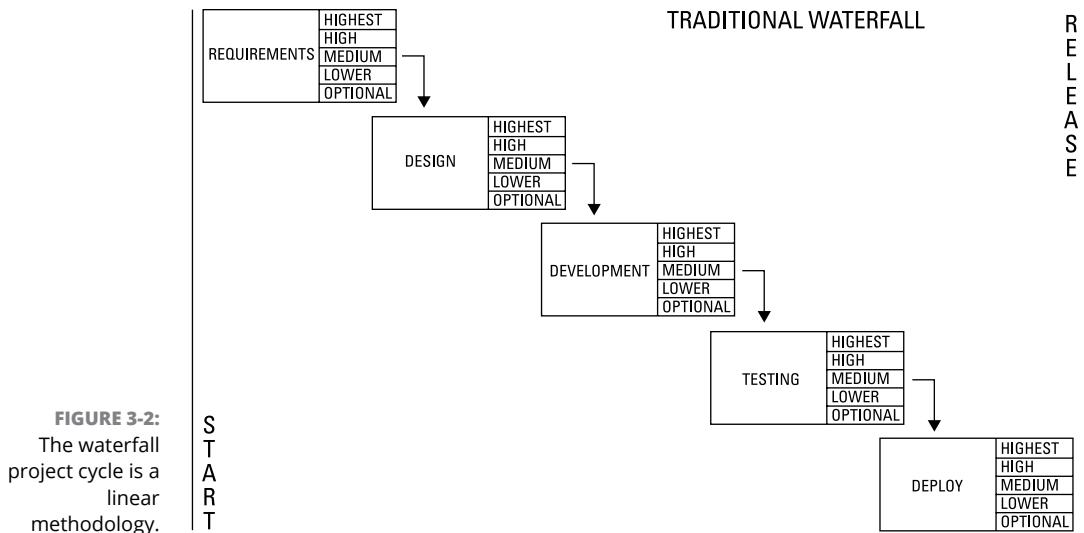
FIGURE 3-1:
A comparison of
historical project
management and
agile concepts.

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.

In the testing phase of a waterfall project, the customer finally gets to start seeing parts of the 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 approaches turn the concept of how product development should be done upside down. Using agile methods, you develop, test, and integrate 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 rather than at the end of development. Development teams identify and remove defects, preventing them from ever finding their way to the customer, just like a gardener can more easily find a weed in a flowerpot than in a cornfield. They not only find and remove the weeds, they prevent the weed seeds from germinating.



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.

Product owner, scrum master, and sprint are terms from *scrum*, a popular agile framework for organizing work and exposing progress. Scrum refers to a rugby huddle, in which rugby players come together to gain possession of the ball. Scrum as an approach, like rugby, encourages the team to work together closely toward a common goal and take responsibility for the result. (You find out more about scrum and other agile techniques in Chapter 5.) We use scrum as an example to explain many of the concepts in the rest of this chapter.

Moreover, with agile product development, 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.

An agile approach to product development will reduce risk during every iteration. In addition, if your product has market value, revenue can be coming in even during development. Now you have a self-funding product!

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 control, and faster failure detection. We describe all these results in this section.

However, these results can't be achieved without a highly competent, enduring, and functional development team. The development team is pivotal to the success of the product. Agile methods emphasize the importance of the support provided to the development team as well as the importance of team members' actions and interactions.



REMEMBER

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

Scrum teams are centered on development teams (which includes feature creators, testers, designers, and anyone else who does the actual work of creating the product), as well as the following two important team members, without which the development team couldn't function:

- » **Product owner:** The *product owner* is a 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:** The *scrum master* 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. They facilitate consensus building and continuous team improvement.

You can find complete descriptions of the product owner, the development team, and the scrum master in Chapter 7. 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 product development offers both greater flexibility and greater stability than traditional projects. First, you find out how agile development offers flexibility, and then we discuss stability.

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

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

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



REMEMBER

The fourth core value in the Agile Manifesto is “Responding to change over following a plan.” Agile frameworks are created with flexibility in mind.

With agile approaches, teams can adapt to new knowledge and new requirements that emerge as development 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 product development teams manage change:

- » At the start of product development, the product owner gathers high-level product requirements from stakeholders and prioritizes them. The product owner doesn’t need all the requirements broken down in detail up front; he or she needs 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 valuable 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.

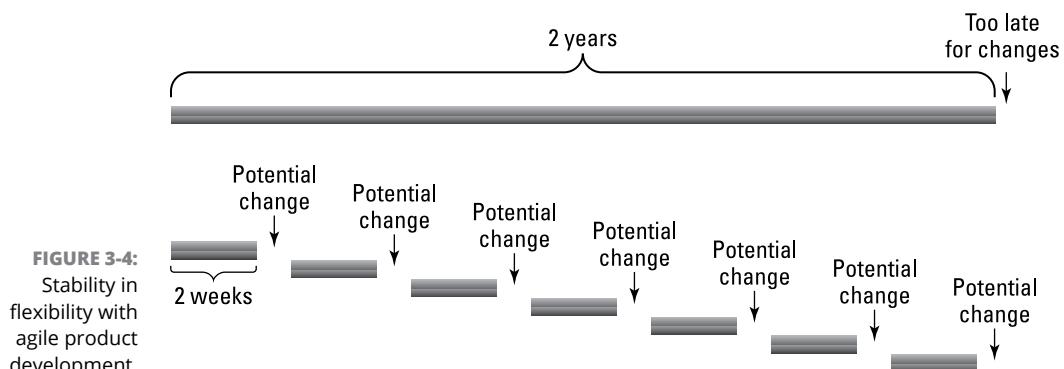
Iterations, or sprints are short — they last up to four weeks, and are often one or two weeks. You can find details about sprints in Chapters 10–12.



TIP

- » 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 next highest-priority requirements.
- » Concentrating on one sprint at a time and on the highest-priority requirements allows the 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 based on changing market or business conditions.
- » The product owner can financially invest in high-priority features first and can choose which features to fund throughout development.
- » 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 development once he or she deems the product has sufficient functionality to fulfill product goals. Agile product development typically ends early, running out of value before it runs out of time or money.

Figure 3-4 illustrates how making changes with agile development 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.



Now look at the bottom image in Figure 3–4. The small steel bars represent a two-week iteration. 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 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 product development is tactically flexible because it is strategically stable. Agile approaches are great at accommodating change because the means for regular change are built into everyday processes. At the same time, iterations offer distinct areas for stability. 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 product development provides a constant cycle of development, feedback, and change, allowing teams the flexibility to create products with only the right features and the stability to be creative.

Reduced nonproductive tasks

At any point in your working day when you’re creating a product, you can work on either 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 develop a product, 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



WARNING

Don't be fooled into thinking that your development team will work only one weekend. If you work one weekend, you'll probably work most weekends from now on. Working overtime sets a false watermark. After you do it, people don't expect less moving forward; if anything, they'll expect more.

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. With agile product development, 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, working with the product owner, then may figure out how to satisfy that need without disrupting the development team.
- » With agile product development, the current 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 16.

Email

Email is not an efficient mode of communication to resolve issues. Product development teams aim to use email sparingly, perhaps when information needs to be sent out or a simple yes-or-no answer is required. Even then, better tools exist, such as persistent chat tools in which the conversation taking place is the same for all participants. The email process is asynchronous and slow: You send an

email, wait for an answer, have another question, and send another email. This process eats up time that could be spent more productively.

Instead of sending emails, 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, product development teams often use the following techniques:



TIP

- » **Demonstrate, don't present.** In other words, show the customer what you've created, rather than describing what you've created. Product development teams always have a shippable increment of their product ready to demonstrate, avoiding the temptation to report hypothetically on work in progress.

Customers and stakeholders can provide even better feedback if the demonstration includes the opportunity for them to experience the product increment for themselves by putting their hands on the keyboard or product themselves.

- » **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. Product development 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, document the decisions made.
- » **Remember that one size doesn't fit all.** You don't have to create the same documents for every development effort. Choose what makes sense for your product, stakeholders, and customers.

- » **Use informal, flexible documentation tools.** Whiteboards, sticky notes, charts, and other visual representations of the work plan are sufficient tools.
- » **Include simple tools that provide adequate information for management about product development progress.** Don't create special progress reports, such as extensive status reports, for the sake of reporting. Product development teams use visual charts, such as burndown charts, to readily convey status while remembering that "working software (or product) is the primary measure of progress" (Principle 7).

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 product development is designed to deliver high-quality, shippable functionality quickly. Agile product development achieves 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 product development.
- » 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 — with every code check-in if necessary.
- » 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 product development is the experience of the team members. Compared with traditional approaches such as waterfall, product development 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, scrum master, and 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 team goal, eliminating the time and focus lost by switching context.

- » 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 creating 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 products and different teams are able to adapt around their specific situation, as you see in the discussion of sprint retrospectives in Chapter 12. Here are some ways that teams can continuously improve:

- » Iterative development makes continuous improvement possible because each new iteration involves a fresh start.
- » Because sprints happen over only a week or two, teams can incorporate process changes quickly.
- » A review process called the *retrospective* takes place at the end of each iteration and gives all 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 control

The work goes more quickly with agile development than under waterfall conditions. Elevated productivity helps increase 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.
- » Based on the working functionality you deliver at the end of each sprint, you determine the work for the next sprint according to current knowledge and updated priorities. You're not locked in to priorities set days, weeks, months, or years ago.
- » When the product owner sets priorities for the next sprint, this action has no effect on the current sprint. With agile development, a change in requirements adds no administrative costs or time and doesn't disrupt the current work.
- » Agile techniques make product 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 need to 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 compiled and integrated daily, and working product is demonstrated, at most, every few weeks. From the first sprint to the close of development, every team member knows whether the team is delivering. Up-to-the-minute knowledge and the ability to quickly prioritize make high levels of control possible.

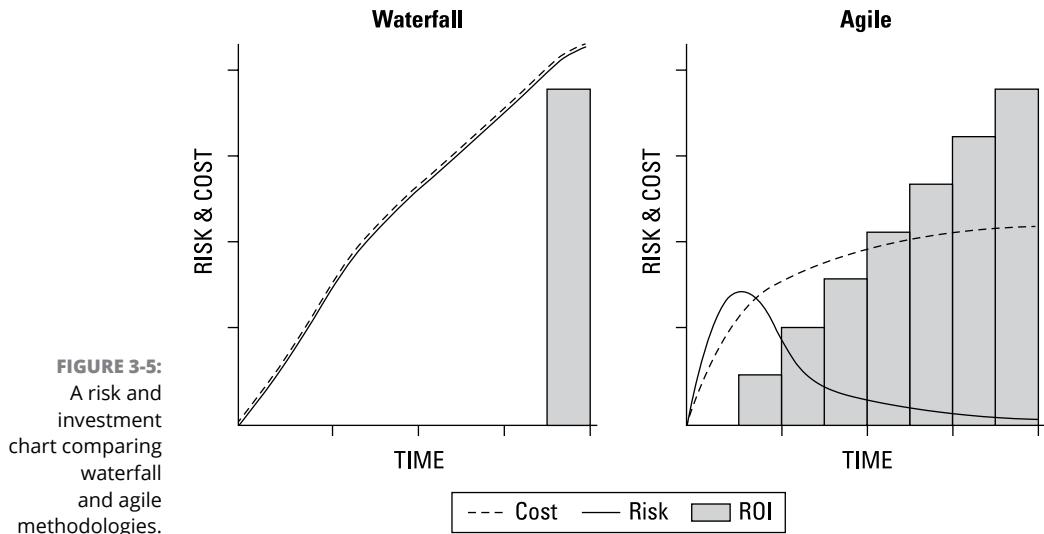
Faster and less costly failure

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.

Along with opportunities for tighter 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 product development in Chapter 17.



Why People Like Being Agile

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

Executives

Agile development 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 product owner and scrum master.
- » Nonproductive efforts are minimized. An 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 development is easier to understand at a glance.

- » Through continuous testing, defects are detected and corrected early.
- » Development 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 project approaches. Speed-to-market advantage is indefensible.
- » **Products can self-fund.** A release of functionality might generate revenue while development of further features is ongoing.

Product development and customers

Customers like agile product development 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 changes in beneficial ways. For example:

- » Agile development creates an opportunity for increased customer satisfaction and return on investment by incorporating change even late in development, nondisruptively.
- » Because the team members and the sprint length remain constant, product 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 development 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 development 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:

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

Management

People in management tend to like agile development 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 up front.

Perhaps you work on product development that does not include software. Technical practices exists for ensuring quality for any type of product. With non-software development, what are ways you can think of to build in quality up front?

Less product and process waste

With agile development, 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 wasted 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 product as soon as the next sprint. As development 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 the 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 with the product owner 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 product 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 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.

IN THIS CHAPTER

- » Knowing your customers
- » Understanding your customer's problems
- » Preventing root causes rather than treating symptoms

Chapter 4

Agility Is about Being Customer Focused

Product development teams build and maintain products that customers need. Understanding who your customer is and the problems he or she needs solved makes product development valuable. In this chapter, you learn how teams work to understand who their customer is, the problem the customer needs solved, and the root causes rather than the symptoms.

Knowing Your Customers

"Who is my customer?" is a fundamental question everyone must ask as they begin product development. Product development teams explore this question frequently, using product and customer usage data to see trends and watch industries and markets closely. Not clearly understanding who your customer is makes product development difficult and rudderless.

Customers range from external paying customers to internal users. Sometimes the customer is even several layers away, spanning wholesalers, distributors, and retailers. Product landscapes also change dramatically and are uncertain. Each new generation of users from baby boomers to millennials brings a new set of needs or considerations. Add to that cultures, races, and ethnicities; new or

changing technologies or inventions; new laws or regulations; and even results from competitive benchmarking. Customers continually demand faster, better, and cheaper. From cars that receive operating system updates every week to wearables to home automation, customers expect more — and more often. Product providers who can meet these demands remain relevant.

Knowing who your customer is will put your product development effort on the right path. But how can you know if you're on the right path? This uncertainty can be unnerving.

Getting comfortable with uncertainty

Modern product development is fraught with uncertainty. In fact, it's the one aspect of product development that's certain. Rarely can we develop the same solution for our customer's ever-changing problems. Product providers continually question, "Have we accurately identified our customer and are we addressing the right problem?" Product development teams need to be comfortable with uncertainty.

Certainty synonyms are predictability, sureness, assurance, and certitude — aspects you would expect of a low-risk investment. Uncertainty is fraught with vagueness, ambiguity, and insecurity — aspects of a high-risk investment. Product development teams continually strive to address the highest risk and highest value for their customers. Throughout their product development journey, customers' needs become more certain as the team learns, grows, and gains experience and opportunity.

With iterative product development, the team can present at every sprint a fully functioning, working product increment — not a half-baked increment — and get feedback. At each sprint, the team aligns the product closer and closer to what the customer needs through a tight, consistent feedback loop. The team becomes more and more certain that what they're creating is what the customer wants. The team validates assumptions along the way, thus reducing uncertainty.

Being comfortable with uncertainty is important because it changes the team's perspective. It enhances the team's curiosity rather than giving them a false sense of assurance. Innovation and better ideas result from a curious team, an attribute shared by all famous inventors.

Uncertainty enables a team to channel its vagueness, ambiguity, and risk into a fruitful learning experience with growth and opportunity. Uncertainty motivates the team, which aligns with principle 5, "Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done." Product development is risky business, in a good way.

Common methods for identifying your customer

Although you can identify customers and their needs in many ways, we discuss several popular methods used by product development teams. In the context of agile development, we refer to *customers* as the end users and *clients* as the people paying for the product or service.

Product owners and developers who are experts in techniques for better understanding their customer are valuable to their organizations and teams. Scrum masters who can facilitate using these techniques are likewise valuable. Each technique puts the team on the path towards certainty.

Product canvas

In *Inspired: How to Create Tech Products Customers Love* (Audible Studios), Marty Cagan wrote, “discover a product that is valuable, usable and feasible.” What he meant by this, as shown in Figure 4-1, is that successful product development hits the sweet spot at the cross-section of

- » **Valuable:** Will customers buy it?
- » **Usable:** Do customers need it?
- » **Feasible:** Can we do it?

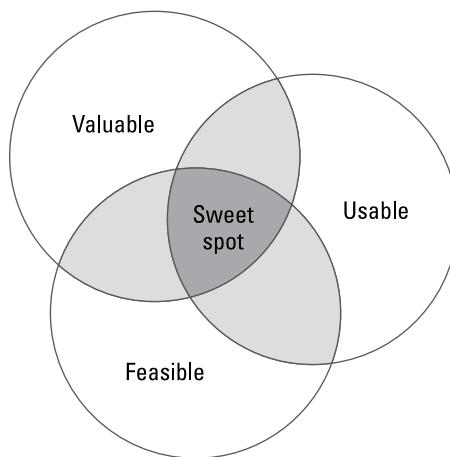


FIGURE 4-1:
The sweet spot
where a product
is valuable,
feasible, and
usable.

Many teams use a visualization technique, such as a product canvas, to start to explore and understand the key factors, partners, unique value proposition,

problems, and potential solutions that may contribute to finding the sweet spot, where value, usability, and feasibility meet.

The product canvas is a collaborative tool that enables teams, in a brief amount of time, to accomplish two tasks: One, start identifying desired goals or product outcomes. Two, validate assumptions about the problem to solve for the customer and ready the team for development. Product canvas exercises can inspire a clear product vision. You learn more about product vision in Chapter 9.

Teams use the product canvas as a visualization activity to create a shared understanding of the customer and his or her needs. The tool serves as a starting point for the team's assumptions and enables them to dive deep in gathering new product insights.



TIP

Product development teams create a shared understanding through visualization. The most effective communication medium for them is face-to-face with a whiteboard, flipchart, or other holistic surface. Why? Whiteboards allow people to not only explain but also draw what they mean. Peter Lencioni, author of “The Five Dysfunctions of a Team” said, “If you could get all the people in an organization rowing in the same direction, you could dominate any industry, in any market, against any competition, at any time.” A shared understanding helps teams row in the same direction.

Many variations of canvases are available, such as a lean canvas or a business opportunity canvas. They all serve a similar purpose of organizing ideas, challenging assumptions, and collaborating to find a strategic direction. Figure 4-2 demonstrates the product canvas we often use with agile product teams. The left half addresses market and customer issues, while the right half addresses product and business issues. The left half defines the customer segment, customer problems with alternatives, the value proposition, channels, and revenue projections. The right half defines the solution, key stakeholders, success factors, resources, partners, and costs. Both halves enable a team to do a more detailed evaluation of their product.

Following are some categories a team might use in building its product canvas:

- » **Customer segment:** Describe the target customer segment that needs the problem solved. For whom is the value being created?
- » **Early adopter:** Describe the initial target customer. Remember, your product can't be everything to everybody — at least not at first. What market segments are opportunities for testing your product idea first?
- » **Problem:** Describe the primary problem experienced by the target customer segment.

Product Canvas™

Customer Segment For whom are we solving a problem? For whom are we creating value?	Problem What is the top problems faced by our target Customer Segment?	Unique Value Proposition How are we uniquely going to solve our customer's problems or satisfy their needs? A single, clear, compelling message that states why you are different and worth buying.	Solution What are the primary ways we are going to solve the problem faced by our Customer Segment?	Key Success Factors How will we measure success? What key metrics are we trying to move?
	Existing Alternatives Who is a potential early user of the solution?	Channels How are they solving the problem today? How will we get (acquire), keep (retain) and grow (sell more to existing) customers? Get/Acquire: How will we drive awareness, interest, activation, usage? Keep/Retain: How will we keep customers coming back? Grow: How will we up-sell/cross-sell customers, encourage referrals?	Key Stakeholders Who are the most important stakeholders whose buy-in we need? Which executives do we need to convince? Who will be our executive champion? Who are the key influencers to these stakeholders? Who else do we need to include in our coalition-of-the-willing?	Key Resources & Partners What are the critical internal and external resources we need to deliver the solution to the customer?
Revenue/Business Value What is the business value of delivering the product/service/capability? (E.g., drive revenue, save money, increase CSAT, competitive differentiator, market positioning, etc.)	Cost Structure What are most important costs inherent in our product model? Which Key Resources are most expensive? Which key activities are most expensive – product development, marketing, customer support?			

FIGURE 4-2:
The product
canvas.

© Shardul Mehta 2015
<http://streetsmartproductmanager.com>



Shardul Mehta

MKT/CUSTOMER

PRODUCT/BUSINESS

- » **Existing alternatives:** Describe available alternatives to your product.
- » **Unique value proposition:** Describe a single, clear, and compelling message that states why or how your product is different and worth buying.
- » **Channels:** Describe the channels for acquiring, retaining, and increasing customers. List ideas for creating awareness, interest, activation, and usage. Describe what will entice customers to return and encourage referrals. List upselling opportunities.
- » **Solution:** Describe the ways the target customer segment's problems will be solved.
- » **Key stakeholders:** List the people who are most important to your product. This list may include people whose buy-in and support you need, executives, and influencers. Who are the people you can trust to criticize your product and tell you the truth?
- » **Key success factors:** Describe how success will be measured — measure outcomes not outputs. Are there key metrics you can use to test your hypotheses?



REMEMBER

Using metrics allows product development teams to evaluate the success or failure of each product release. Metrics help them see if they truly understand and are solving their customer's problems. Product development teams don't consider their work completely done until those objectives are met. The product canvas and other tools outlined in this chapter can help identify these business metrics early. Agile product development techniques increase the likelihood that bad or wrong ideas are quickly discarded.

- » **Key resources and partners:** Describe the critical internal and external people, equipment, or resources needed to deliver the solution to the customer.
- » **Revenue/business value:** Describe the business value of delivering the product, service, or capability. Consider what will drive revenue, save money, increase customer satisfaction, differentiate you from your competition, improve market positioning, and more.
- » **Cost structure:** Describe the important costs inherent in the product model. Identify what will be most expensive, for example, resources, activities, development, marketing, or support.

Using the foundation of a product canvas not only helps the team to better understand the customer and desired outcomes but also enables the product owner to build a concise yet strategic product vision statement with a supporting product roadmap. The product vision statement and roadmap are discussed in Chapter 9.

Customer map

Other customer-focused mapping tools can be powerful visualization activities for a team seeking to better understand its customers. We introduce two common customer mapping tools: a journey map and an empathy map.

A *journey map* helps the team visualize the day-to-day experience a customer goes through when accomplishing a goal or addressing a specific problem. Goals are followed by actions in a timeline format, by calling out the user's emotions or thoughts. The journey map creates a narrative. The insights gained inform product designs. Figure 4-3 outlines the flow of a journey map and the relationships between the customer's goals and his or her steps, insights, and emotions.

An *empathy map* (see Figure 4-4) helps the team think through the user's emotions and senses. It explores what the customer sees, hears, thinks, feels, and does. It identifies pain the customer experiences and how the product can give the customer the gains he or she seeks.

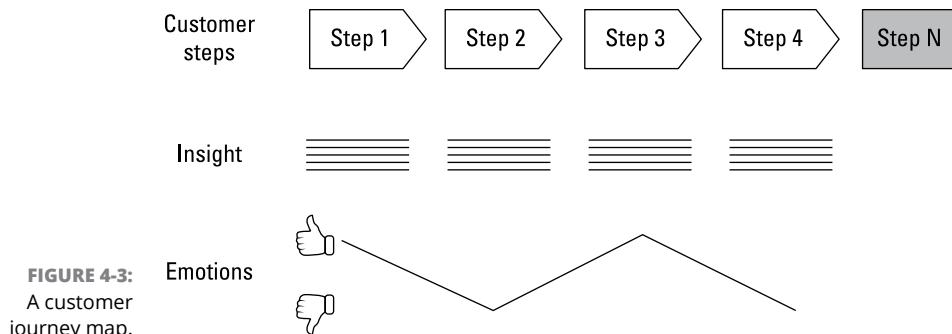


FIGURE 4-3:
A customer
journey map.

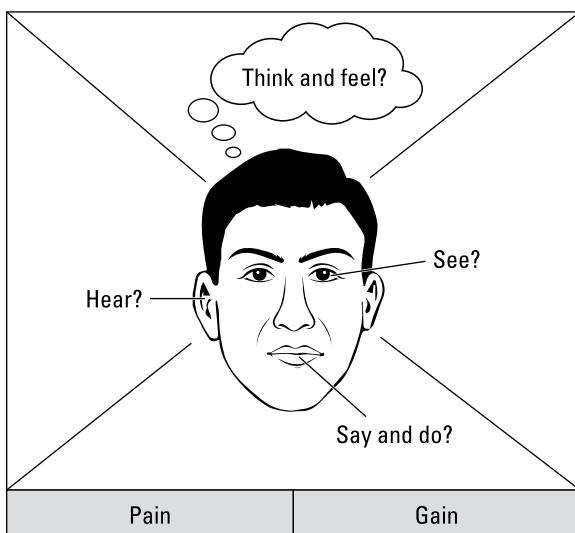


FIGURE 4-4:
A customer
empathy map.

Teams build customer maps together, learning and digging deeper into their customer's needs, motivations, and challenges. They openly discuss their perceptions, observations, and insights to validate their understanding along the way.

Job-to-be-done lens

Jobs theory, developed by Clayton Christensen and the Harvard Business School in response to his overwhelmingly popular theory on disruptive innovation, refers to an approach used to discover functionally, socially, and emotionally why customers make the choices they do. According to the theory, products and services are hired to help the customer make progress. They call the customer's progress the *job-to-be-done*, which when understood enables significant innovation. Beware that if a product can be hired, it can also be fired.

UNDERSTANDING THE JOB-TO-BE-DONE OF A CONDOMINIUM

Clayton Christensen tells the story of a home builder in Detroit in 2006 who sold condominiums to down-sizers — retirees looking to move out of the family home, divorced parents, and single parents. In an effort to increase home sales, the builder considered many options with a focus group. They considered adding a bay window, changing colors, or making structural changes, and then evaluated to see whether their change made a difference in sales. The results were negligible.

The builder decided to interview his previous customers to learn the reasons and timeline that led to their condo purchase. Unfortunately, no patterns emerged except one: Every buyer mentioned a concern about their dining room table. Confused at first, the builder realized that while the table was somewhat inexpensive, well-used and possibly dated, it was central to the family's activities: homework, birthdays, holidays, and other gatherings. The dining room table represented their family.

Buyers were hesitant about buying condos not because of the physical structure but because of the anxiety associated with having to give up something that had profound meaning. The job-to-be-done was to enable buyers to maintain family gatherings, memories, and traditions even when downsizing.

In response, condominiums were redesigned with a larger space for the dining room table and each new buyer was given a two-year storage unit with a sorting area for their belongings. Christensen cited that these changes allowed the builder to not only differentiate himself from his competitors but also raise his price. Even better, in an industry experiencing a severe economic decline at the time, his business grew by 25 percent. Understanding the job-to-be-done made all the difference.

The jobs-to-be-done approach adheres to four principles:

- » **Job is shorthand for what an individual really seeks to accomplish in a given circumstance.** Product development teams need to understand the experience the customer is trying to create, which is not a straight-forward task.
- » **The circumstances are more important than customer characteristics, product attributes, new technologies, and trends.** This principle speaks to seeing the innovation through the lens of the customer's circumstances.
- » **Good innovations solve problems that formerly had only inadequate solutions — or no solution.** If customers see only two options, a third option

that addresses all the relevant criteria can help fickle observers become customers.

» **Jobs are never simply about function — they have powerful social and emotional dimensions.** Understanding the social and emotional dimensions of a customer's choice makes all the difference in the customer's purchasing decisions.

Christensen shares how viewing products through the jobs-to-be-done lens significantly increased sales of his sample populations. Specific examples he cites include milk shakes, condominiums, and even Reese's peanut butter cups.

Product development teams use the job-to-be-done theory throughout their product development, particularly as they evaluate the timeline from when customers identify a need to their purchase. Figure 4-5 shows an example job-to-be-done timeline. Close interactions with stakeholders and customers help validate their assumptions.

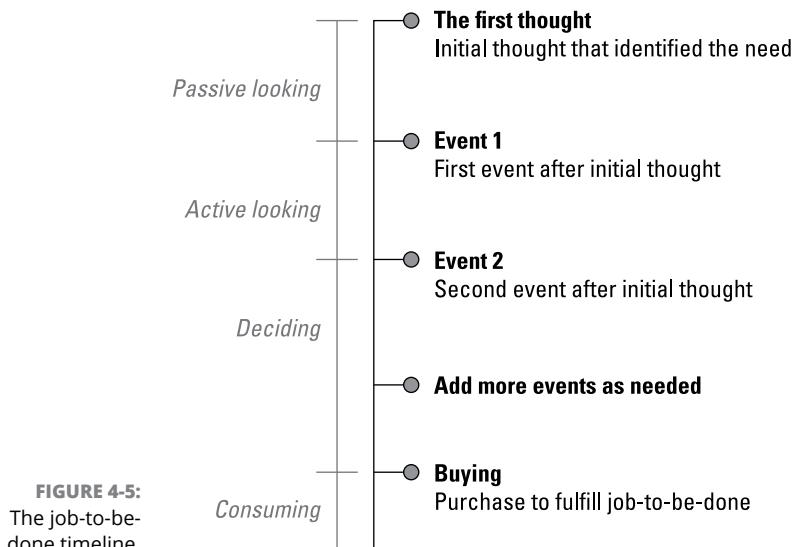


FIGURE 4-5:
The job-to-be-done timeline.

Product development teams can benefit by considering the job their product or service must fulfill for the customers. Better understanding the job-to-be-done requires customer interviews so you can truly understand their needs.

Customer interviews

What better way to understand a customer's needs than to talk to the customer? Teams interview customers because they value "customer collaboration over contract negotiation" and "individuals and interactions over processes and tools." Customer interviews are critical for enabling progressive elaboration techniques such as decomposition and story mapping. See Chapter 9 to learn about progressive elaboration.

The key for performing a successful interview is to get customers to talk by allowing them to tell stories about their problems and imagine possible solutions. You escape from your perspective into the customer's. Interviewees will discover things they didn't know and interviewers will discover whether they're making wrong assumptions. Consider the interview as an opportunity to validate the team's assumptions and to vet both good and bad ideas.

Table 4-1 outlines the types of questions interviewers should and shouldn't ask.

The types of question on the left invite storytelling. They encourage the customer to answer beyond any biases you may have. The examples on the right give significant constraints to customers on how they can answer the question, and don't encourage much conversation or storytelling. Instead, get them talking.

TABLE 4-1 Customer Interview Do's and Don'ts

Do This	Don't Do This
Use open ended questions beginning with why, how, and what. Follow up with more questions such as, "Tell me more."	Start with a pitch or description of your product idea.
Use questions that invite storytelling.	Multiple choice or one word/one phrase responses.
Tell me about...	Do you like...? Do you want...?
How do you know...?	Would you use this?
Can you give me an example of...?	What do you think of our product?
What do you wish you could do...?	What requirement should we add?
What do you know about...?	How often do you...?
How do you feel about...?	Would you ever use...?
How would life be different with...?	Would you pay money for...?
When was the last time you...?	How often would you...if you could...?
What happened when you...?	Take vague terms like "difficult," "expensive," "complex" at face value.
Ask about past behavior, rather than hypothetical/ideal	
What do you mean by...?	

Product discovery workshop

Product development teams use *product discovery workshops* to gather product ideas and insights from stakeholders and scrum team members. These workshops have many different formats and applications.

Workshops, if done correctly, allow creative juices to flow. What is key is creating a safe environment where ideas can be shared openly. Timeboxes (setting a time limit on the discussion) for topics are helpful, as well as plenty of Post-it Notes and markers, and the freedom to diverge, converge, explore, and discover.

Some people, especially introverts, find that receiving prepared agendas beforehand is helpful for getting their thoughts flowing. Agendas can also help everyone to arrive prepared and ensure that the workshop objectives are accomplished. Be careful to not use the agenda to provide excess structure, however. Lightweight structure, particularly with workshops, can help the discussion go where it needs to go.

The frequency and participants of the workshops depend on the team, the product, and the problem to solve.

Figuring Out the Problem Your Customer Needs to Solve

The solution to a customer's problem may not be what you think it should be. It may not even be what the customer thinks it should be at first. It may not be what the customer tells you he or she wants. The customer paying you to develop the product and end-users may even have conflicting views of what's needed at first. One of the most difficult things about product development is that customers often don't know what they want until they interact with it.

Clearly understanding who your customer is makes understanding their problems much easier. Teams understand this and take the time to understand customer problems and their root causes. Addressing symptoms can be costly. Charles Kettering, owner of 186 patents, said, "A problem well-stated is half-solved." Several methods exist for digging deeper into customer problems and their root causes.

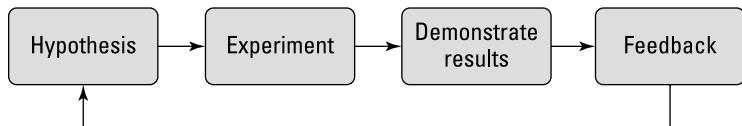
Using the scientific method

Central to agile product development is the scientific method. *The scientific method*, which has characterized natural science since the seventeenth century, consists of systematic observation, measurement, and experiment, and formulating, testing,

and modifying hypotheses. Product development teams ask questions to form hypotheses, test those hypotheses, and then evaluate the results, over and over again.

Teams use the scientific method to move their understanding of customer needs and problems from uncertainty to certain, as shown in Figure 4-6.

FIGURE 4-6:
The scientific
method.



To use the scientific method, the team begins by making observations or asking questions about a particular customer problem. The team forms a hypothesis about that problem. Then it discusses possible solutions in the form of requirements to test the hypothesis. Each requirement identifies the desired outcomes to be measured so the team can evaluate the results.

The team builds the product increment during the sprint and gathers feedback at the sprint review. Feedback is prioritized against the backlog. Product owners release the increment when they feel enough value is created and the requirement is ready to be tested in the wild. The team then positions itself for capturing more customer feedback. The cycle repeats as necessary. Every product backlog item should achieve a desired outcome.



TIP

Teams use the scientific method not only for product development but also to improve the development process during retrospectives. At each sprint retrospective, they form a new hypothesis for improving, test the hypothesis with an experiment during the sprint, and then evaluate the results — another example of how the scientific method is central to agility.

THE SCIENTIFIC METHOD AND R&D

Research and development clients have a hypothesis based on a set of premises. Each sprint empirically validates or invalidates that premise as they collect feedback from subject matter experts who critique the empirical findings. If the validation is weak, the scrum team iterates on that premise until it is defensible. If the validation is strong, the team progresses to the next premise. At the end, the team writes a white paper or journal article saying “This hypothesis is either valid or invalid and here is how we empirically know.”

Failing early is a form of success

The mantra “fail early, fail often, but always fail forward” as stated by John C. Maxwell motivates teams to accelerate their learning through continuous and early delivery of value. In other words, try a few inexpensive experiments and then evaluate the results before investing more. This allows the team to spend the least amount of time and effort on solutions that may be thrown away. Understanding the problems customers need solved can take a few attempts. Product development teams embrace failure as learning, especially when they learn inexpensively.



REMEMBER

It's not a failure when you learn something after only a week's worth of effort!

With agile approaches to product discovery, invalidating assumptions is as good as validating assumptions. The goal is to learn early what does and doesn't work before a larger investment is made.



REMEMBER

The term *minimum viable product (MVP)* is a concept introduced in lean startup by Eric Ries to help teams learn through failure quickly and cheaply. The point is to quit trying to create everything before you deliver and test anything. Product owners and development teams look for the next minimum viable product increment needed to test and validate whether their solution is viable. Agile Principle 10: “Simplicity — the art of maximizing the amount of work not done — is essential.”

NORDSTROM INNOVATION LAB

Nordstrom used a team called Innovation Lab to experiment with various product ideas in its stores. During the course of a week, the team would run a sprint to prove an idea by creating a product based on rapid feedback loops directly with customers in the store.

In one case, the team created an iPad app for using photos to compare sunglasses while choosing sunglasses to purchase. Every time the development team implemented a new requirement, it would give the app to a sales rep to use with a customer, and then gather feedback right away about how the customer used it or liked it. The team scrapped some requirements customers didn't like and, based on feedback, created requirements the team hadn't thought of in its planning.

At the time, the customers and the staff liked the product. However, the rest of the story is what's interesting. The product is not used at Nordstrom's today. Why? We don't know. Does that mean it was a failure? We don't think so. If they wasted any time on developing the product, it was only one week. They failed cheap, which freed their team to work on the next, more valuable problem.

Defining customer-focused business goals

Agile product development is purpose driven, that is; it always begins by clearly defining and understanding the desired outcomes for the customer. Every release, sprint, requirement, and task is planned with a specific outcome in mind. Establishing goals in customer terms enables a team to keep its focus on the customer, maintain strategic stability, and reserve tactical flexibility to accomplish the problem to be solved.



TIP

Product development teams achieve desired outcomes. They continually validate that their ladder is up against the right wall. Creating more of a bad product faster is not a good idea.

Several goal formats are available to help a team focus on the customer. The most effective goal statements are simple, emotionally engaging, motivating, and memorable. Product development teams discuss the goal every day during their daily scrum.



REMEMBER

US President John F. Kennedy communicated the following goal in 1961: “The US should commit itself to achieving the goal, before this decade is out, of landing a man on the Moon and returning him safely to the Earth.” The goal was brief, emotionally engaging, motivating, and powerful! Obtaining this goal was the desired outcome that drove product development to follow.

Product release goals begin with “Enable my customer to . . .” Iteration or sprint goals begin with “Demonstrate the ability to . . .” Even at the individual requirement level (sometimes referred to as a user story), you describe the requirement using goal statements because they are succinct, define the target audience, and define what benefit will be received and why. Regardless of the specific pattern used, it’s important to keep the desired outcomes for the customer front and center. (Read Chapter 10 to learn more about release planning, sprint planning, and user stories.)

In this way, everything a team does throughout the sprint benefits the customer and solves his or her problem in some way. Each task is more meaningful, and each roadblock must be overcome. High-performing teams use goals to accomplish business outcomes iteratively, step by step, learning by learning.

Goals should be SMART, or Specific, Measurable, Acceptable, Realistic, and Time-bound, as attributed to Peter Drucker’s Management by Objectives in his 1954 book *The Practice of Management*.

Story mapping

Story mapping, made popular by Jeff Patton and Peter Economy in their book *User Story Mapping* (O'Reilly Media, Inc.), is another visualization activity used by product development teams. Similar to a journey map, a *story map* helps a team gain a shared understanding of the process or journey users and customers need their product to take as well as the various alternatives for iteratively improving the experience. The story map is organized to clarify for everyone the minimum viable product as well as a path for future releases. Teams can see a more holistic view of how their ideas fit into the overall user experience.

For example, Figure 4-7 outlines how a user might submit a transaction dispute if he or she were charged incorrectly while using a mobile banking application. The sequence with the least amount of functionality to accomplish the desired outcome is across the first row. Other alternatives for eventually improving the user's experience at each step are prioritized vertically.

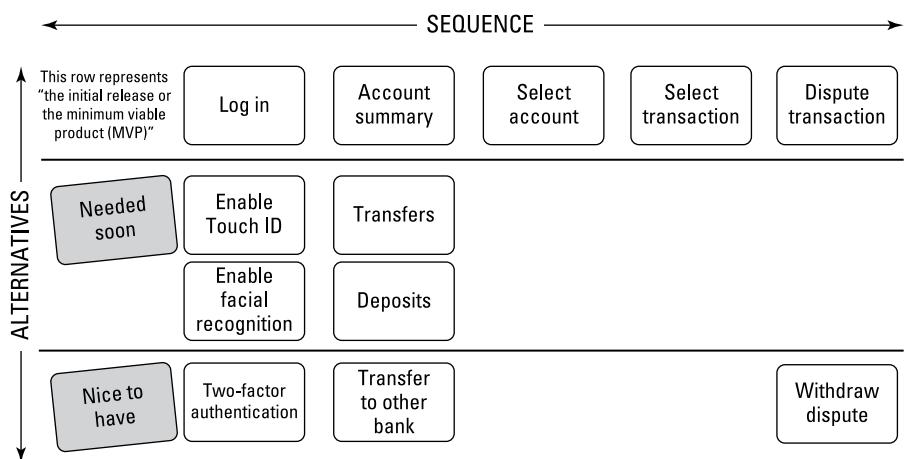


FIGURE 4-7:
A user story map.

By mapping and visualizing the process as a team, the individual requirements make a lot more sense. The team can draft the product backlog together, creating a cohesive team-defined solution based on the customer's experience.

Liberating structures — simple rules to unleash a culture of innovation

The last but certainly not least approach we discuss is liberating structures, which are a set of microstructures compiled and shared by Keith McCandless and Henri Lipmanowicz. *Liberating structures* (www.liberatingstructures.com) provide

alternative ways of structuring and facilitating collaboration. They enable liberation of content and ideas through minimally structured interactions and are designed to help people understand and solve complex problems.



TECHNICAL STUFF

Macrostructures refer to organizational policies and processes that constrain microstructure activities. Conventional (non-liberating) macrostructures include activities such as traditional presentations, discussions, brainstorms, and reporting.

A menu of dozens of liberating structures defines an invitation for participation, how to arrange the physical space and materials needed, how participation is distributed, how groups are configured, and a sequence of steps and timeboxes. Each liberating structure can be used for brainstorming, discussing, and discovering solutions to complex and challenging problems.

An example of a liberating structure often used by product development teams is *1-2-4-All*, which is designed to engage everyone in the workshop to simultaneously generate questions, ideas, and suggestions. The exercise begins with a one-minute silent self-reflection on a shared challenge, framed as a question. Next, pairs are formed and spend two minutes building on the ideas. Then groups of four spend four minutes noting the differences and similarities and developing new ideas. Lastly, each group conducts a five-minute presentation of one important idea that stood out to them. A bell or timer helps everyone work within their timebox. The cycle can be repeated, if needed. In a few minutes, the group is able to generate more and better ideas harnessing the mind-power of their collective group.

Liberating structures are perfect for the types of problems agile product development teams are solving for customers every day.

Understanding Root Cause Analysis

Root cause analysis (RCA) is a systematic approach for identifying root causes of problems or events as well as an approach for responding to them. It's based on the basic idea that effective management requires that you not just deal with symptoms of problems but also find a way to prevent problems.

RCA is a critical aspect to consider when evaluating the problem your customer is trying to solve. Like physicians, product development teams strive to address the problem at the root rather than symptoms. When the root is addressed, or even prevented, the symptoms disappear. Addressing the root cause can be more challenging but often yields a more elegant and simple solution.

Following are the typical steps teams follow as they perform root cause analysis:

1. **Define the problem.** Collaboratively create a problem statement.
2. **Collect data.** Assemble data to support the problem.
3. **Identify possible causal factors.** List factors that cause the problem from the gathered data.
4. **Identify the root causes.** List the root causes.
5. **Recommend and implement solutions.** Form a hypothesis to test.



TIP

RCA is extremely useful not only to product owners and developers when solving customer-focused problems, but also to scrum masters as they work to remove impediments or teach the team to do so. Impediments need to be resolved at the root so that they never raise their ugly head again. Scrum masters resolve impediments not only tactically (remove reactively) but also strategically (prevent proactively).

At any time, product development teams may use several approaches to better understand root causes of the problems they're trying to solve. In this section, we discuss three approaches:

- » **The Pareto (80/20) rule:** If 20 percent of root causes are addressed, they can positively benefit the remaining 80 percent.
- » **Five Why's:** A problem can be debated five generations or layers back to discover the root cause.
- » **Ishikawa (or Fishbone diagram):** A problem can be evaluated across multiple categories — typically people, process, tools, and culture — to expose root causes.

Pareto rule

The Pareto rule, named after economist Vilfredo Pareto (1985), also known as the 80/20 rule, specifies that 80 percent of the effects come from 20 percent of the causes. This rule is a reminder that the relationship between inputs and outputs is not balanced.

Teams use the Pareto rule as part of root cause analysis as they gather data from help desk incidents or customer complaints, for example. From the data, the team is able to categorize 20 percent of the top failures as affecting 80 percent of the

reported incidents. Consider a few other applications of the Pareto rule to product development:

- » 80 percent of the complaints come from 20 percent of your customers.
- » 80 percent of the product's functionality comes from 20 percent of the developer's effort.
- » 80 percent of profits come from 20 percent of product customers.
- » Of all the apps on a mobile phone, users frequently use about 20 percent.
- » Getting the 20 percent right yields a higher return and solves root causes, reducing symptoms.



TIP

The Pareto rule is often used by product owners as they manage their product backlog. According to the rule, 20 percent of the backlog will create 80 percent of the value for the customer. Prioritize the top 20 percent first. Then reevaluate, but be ready to move on to the next investment idea after the 20 percent is met. Agile projects run out of value before they run out of time or money.

Five why's

The *five why's*, also known as the *nine why's* in liberating structures, is a method that uses a series of *why* questions to drill down into successive layers of a problem. The basic idea is that each time you ask why, the answer becomes the basis of the next why. It's a simple tool that is useful for less complex problems.

Mind maps, which start with a central problem statement, can be helpful for guiding the group or team through this analysis. As the group reaches the fifth (or ninth) why, you can be more certain that the group is nearing a root cause.

One application of this technique is to more deeply analyze the results of a Pareto analysis. Here's an example of how to use the five why's with the same mobile banking app we considered earlier in this chapter:

Problem: Customers complain about excessive overdraft fees.

Why 1: Why are customers complaining about excessive overdraft fees?

Answer 1: Customers are not notified that their account is overdrawn.

Why 2: Why are customers not notified about overdrafts?

Answer 2: An indicator or alert is not provided to the customer when an overdraft exists.

Why 3: Why does the customer not see an indicator or receive an alert when an overdraft exists?

Answer 3: Bank policy requires a physical letter be sent when an overdraft exists.

Why 4: Why does bank policy require a physical overdraft letter be sent to customers?

Answer 4: The new account contract, signed by the customer, stipulates a physical letter.

Why 5: Why does the new account contract stipulate overdraft letters will be mailed?

Answer 5: The mobile app did not exist when the new account contracts were created.

Of course, you may need to ask “Why?” more than five times to solve the problem. The point is to peel away surface-level issues to get to the root cause. Five why’s can be used whenever the team is debating a problem or needs to better understand a problem with either the product or the development process.

Ishikawa (fishbone)

An *Ishikawa diagram*, or *fishbone diagram*, is a causal diagram created by Kaoru Ishikawa in 1968 that shows the causes of a specific event. Ishikawa diagrams sort possible causes into various categories that branch from the original problem, as shown in Figure 4-8. Also called a cause-and-effect diagram, an Ishikawa diagram may have multiple sub-causes branching off each identified category.

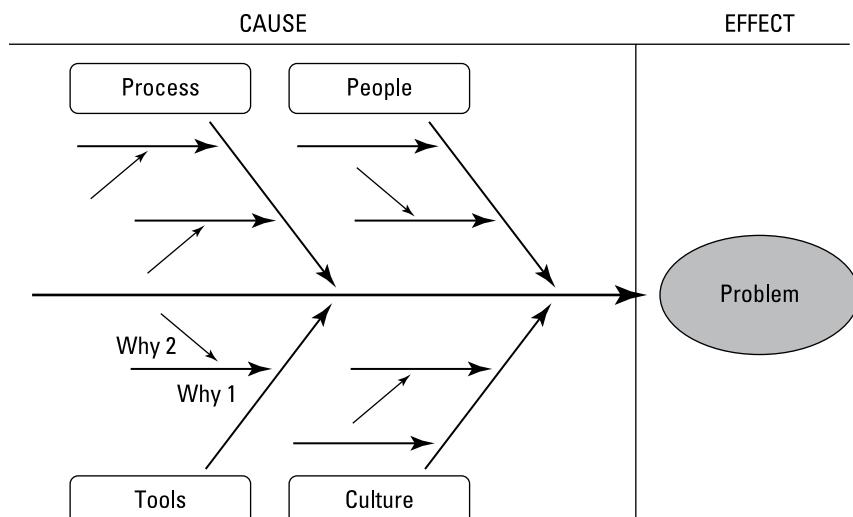


FIGURE 4-8:
The Ishikawa diagram.

To begin, teams align on a single problem statement. They choose wording that everyone can agree to. Multiple iterations of the problem statement may be required until everyone is comfortable.

After the problem statement is clear, the team chooses categories for the spine of the fishbone. Categories can be whatever the team believes will be helpful, such as analyzing the problem across people, process, culture, and tools. Other helpful categories are materials, environment, management, product line, country, and state.

Combining the Ishikawa diagram with a five why's analysis can help trace each fishbone to its root causes. Note that the analysis often results in the same root causes mentioned multiple times throughout. This is a sure sign that you're on the right track to truly understanding the problem rather than a symptom.

In this chapter, we discussed how understanding customers and their problems are central to agility. Forgetting them leads to waste and regret. Remembering them leads to innovation and success. Although identifying customers and their problems can be difficult to grasp, you've learned several techniques to guide your team conversations to move your uncertain product iteratively closer towards certainty.

With root causes understood, teams, like physicians, can address problems rather than symptoms. Various agile frameworks are available for helping product development teams ferret out root causes. Those frameworks are described in the next chapter.



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 for both virtually distributed and physically collocated 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.

Discover the advantages of small, self-organizing, self-managing, long-lived, and enduring “permanent” teams.

IN THIS CHAPTER

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

Chapter 5

Agile Approaches

In previous chapters, you read about the history of agile product 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 agile approaches used today.

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 agile product development, 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.

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

- » Demonstrating valuable and potentially shippable functionality in short iterations called *iterative development*
- » Emphasis on simplicity, transparency, and situation-specific strategies

- » Cross-functional, self-organizing teams
- » Working functionality as the measure of progress
- » Responsiveness to changing requirements

Synonyms of the word *agile* — including resilient, flexible, nimble, adaptive, lightweight, and responsive — give additional insights into what it means to be agile.

Similarly, throughout this book, we reference agile teams. *Agile teams*, which include scrum teams (scrum being the most popular agile framework), are teams who adhere to the agile values and principles to become more resilient, flexible, nimble, adaptive, lightweight, and responsive in meeting their customer's needs.

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

With regards to product development, an 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 commonly used: 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 specific 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.)

Most organizations we work with that are using an agile approach for product development are 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 progress; and using extreme programming practices to frontload quality. 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 product development 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 5-1.) 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.

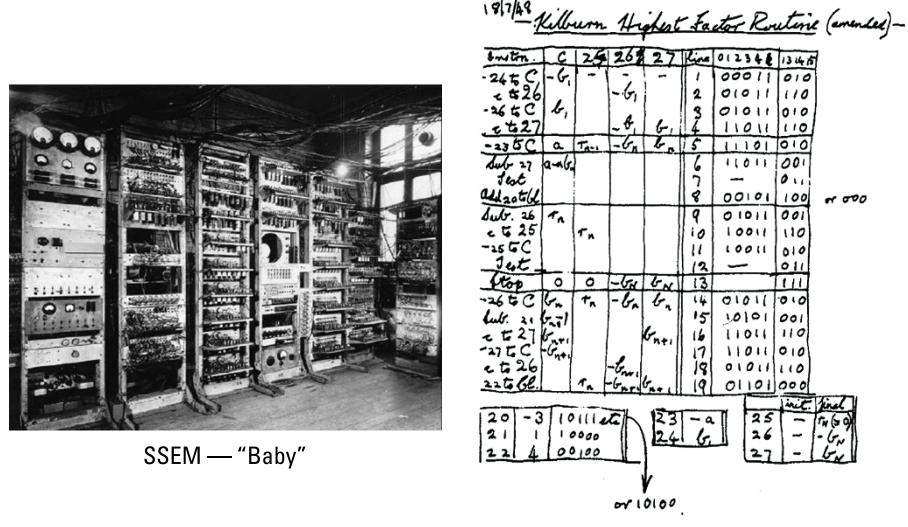


FIGURE 5-1:
Early hardware
and software.

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 5-2, you can see where that name came from.

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.

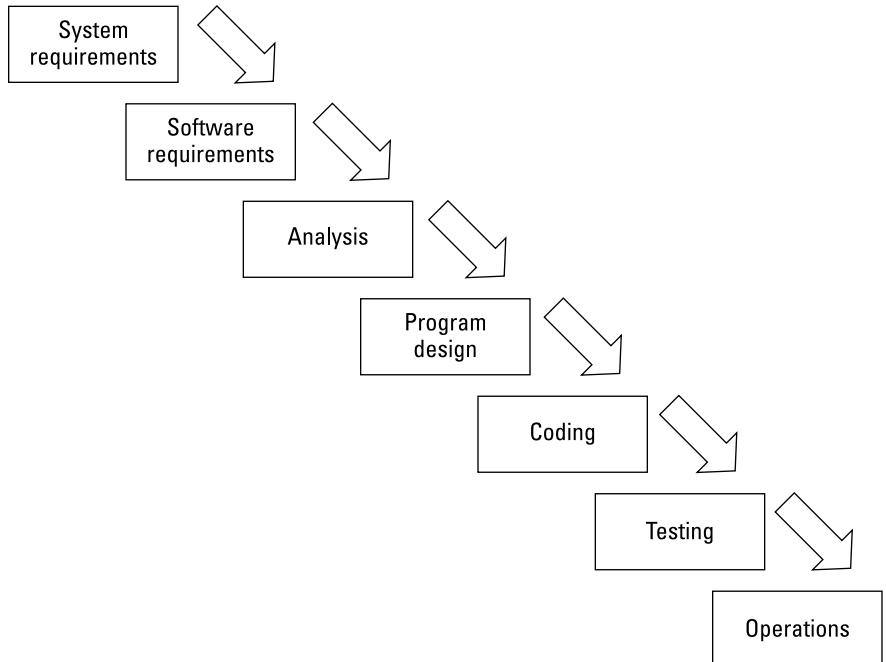


FIGURE 5-2:
The origins of
waterfall.

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 5-3 to illustrate that iteration.

Now, we're not sure if the page with this diagram was stuck with chewing gum to other pages, but the software development community by and large lost this part of the story. Historically, product development in general has been constrained by this linear and defined process control thinking. By embracing the idea that you might not know everything when you first start developing a component and will probably have to revisit your assumption of what is the right thing to ensure that it becomes what your customer needs, you take the first step to becoming more agile. Agile techniques might have come to prominence 40 years earlier if people had taken Dr. Royce's advice to heart!

This story is specific to technology product development, but iterative, empirical process controls apply to non-software products as well.

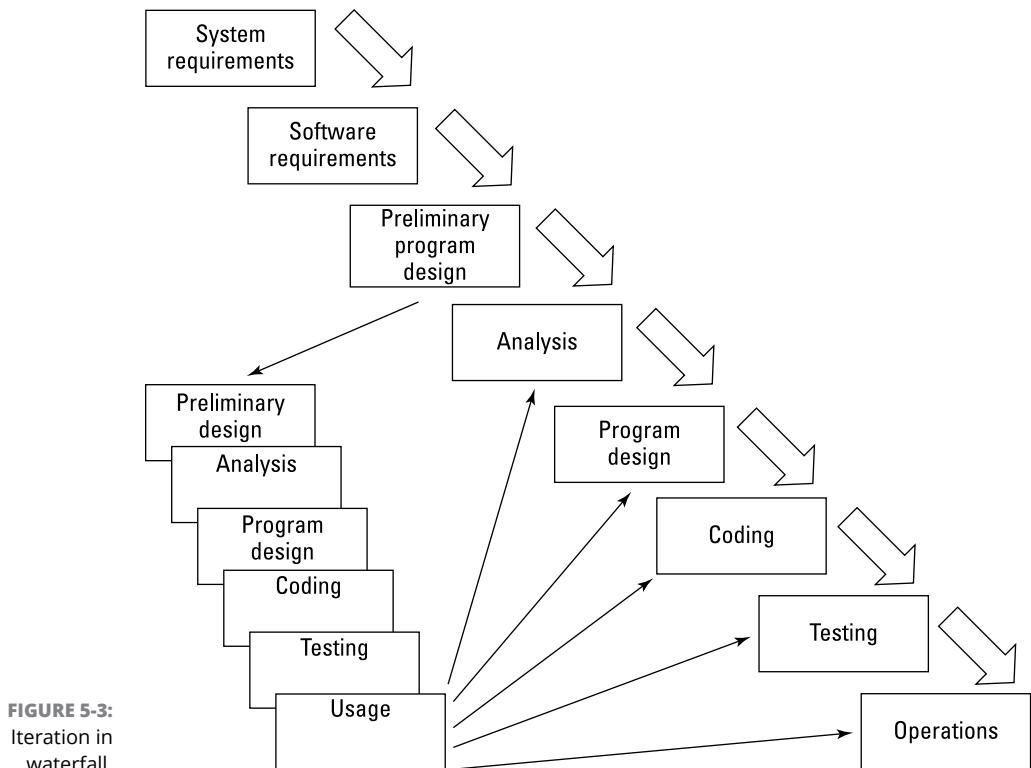


FIGURE 5-3:
Iteration in waterfall.

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 to product development: 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 Japan in the 1940s, 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 product 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 product 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 maximizing 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 objectives) — 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.
- » **Decide 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 by experimenting with hypotheses to find solutions. We discuss the scientific method in more detail in Chapter 4.
- » **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 in integrity (quality).** 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. Optimizing only one part will sub-optimize the whole system. Solve problems, not just symptoms. Continually pay attention to bottlenecks throughout the flow of work and remove them. Think long-term when creating solutions.

Understanding kanban

Beyond the lean principles, one of the most common lean approaches used by product development 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 policies explicit.
- » Implement feedback loops.
- » Improve collaboratively, evolve experimentally (using models and the scientific method).

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 product development 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 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 a Japanese word that has various meanings depending on which alphabet it's written in, but in essence it means *sign*, *large visual board*, *signal cards*, or *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. Product development teams use kanban boards or task boards to expose their progress and manage their flow of work (described in more detail in Chapters 6 and 11).

» **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 done and receiving feedback on what has been 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 product development 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, product development work items move more slowly if development team members 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 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 acceptance in the queue to completion. The team knows the cycle time by tracking the time from when work begins to when it is completed. The team optimizes flow by identifying and removing bottlenecks that keep its lead and cycle times from decreasing.

The foundational principles of kanban enable its effective use with other frameworks such as scrum. These principles include the following:

- » Start with what you do now.
- » Agree to pursue evolutionary change.
- » Initially, respect existing roles, responsibilities, and job titles.
- » Encourage acts of leadership at all levels in your organization — from individual contributors to senior management.



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 product because doing so 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 according to Digital.ai's 14th annual *State of Agile Report 2020*, 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 transparency, inspection, and adaptation throughout development. The scrum approach is shown in Figure 5-4.

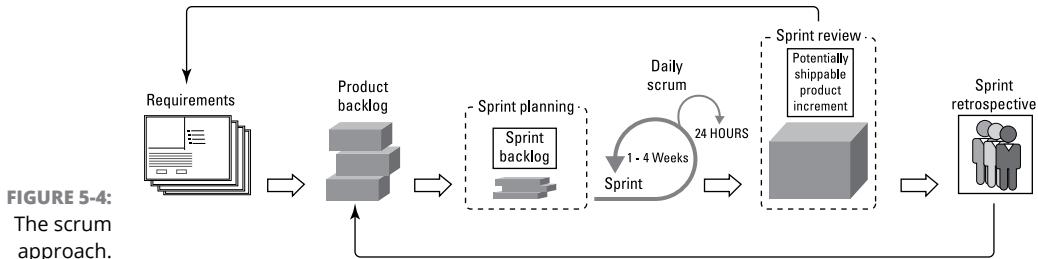


FIGURE 5-4:
The scrum approach.

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 demonstrated for stakeholder feedback. Based on this feedback, the product owner determines the next steps, whether to release the increment, and what adjustments to make to the product backlog moving forward.

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 5-5.

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

- » During a sprint, you make all progress transparent and conduct *constant inspections* to assess progress toward the sprint goal, and consequentially, toward the release goal.

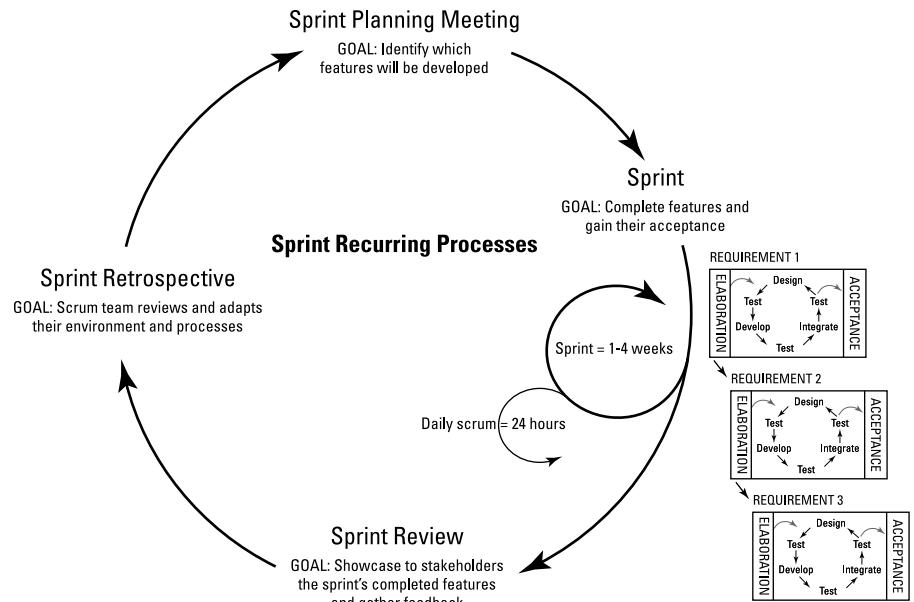


FIGURE 5-5:
Sprints are
recurring
processes.

- » To organize the day, you hold a *daily scrum meeting* by coordinating what the team will work on today. Essentially, the scrum team inspects its progress toward the sprint goal and adjusts its plan to achieve the sprint goal based on the reality of the day.
- » At the end of the sprint, you use a *sprint review meeting* and a *sprint retrospective meeting* to assess product improvements and team performance, respectively, 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 be different than the problem you'll need to solve in the future anyway.

Understanding scrum roles, artifacts, and events

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

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

- » **Product owner:** Represents and speaks for the business needs of the product.
- » **Development team:** Performs the day-to-day technical implementation work. The development team is dedicated to the product and each team

member is *multi-skilled* — that is, although team members may have certain strengths, each member is capable of doing multiple product development jobs.

- » **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 product. Although stakeholders are not official scrum roles, it is essential for scrum teams and stakeholders to work closely together throughout development.
- » **Agile mentor:** Someone who has experience implementing agile principles, practices, and techniques, and can share that experience with a team. Sometimes called an *agile coach*. Often this person is external to the product'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 *artifacts*, which are tangible deliverables that the scrum team makes transparent and uses to continuously inspect and adapt:

- » **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 is fluid throughout the product lifecycle. 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 enabling the team to achieve a specific sprint goal. 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, sprint backlog tasks can be changed only by the development team as it sees fit to ensure that it achieves the sprint goal.
- » **Product increment:** The usable, potentially shippable functionality. Within the context of a single sprint, the product increment includes functionality from requirements that have been elaborated, designed, developed, tested, integrated, documented, and approved to meet the business needs for which it was intended. Whether the product is a website or a new house, the product increment should be complete enough to demonstrate its working

functionality. Product increments are released to the customer after enough shippable functionality has been demonstrated to meet the customer's business goals. In other words, it may take more than one sprint to generate enough valuable functionality to ship to the customer.

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.
- » **Sprint planning:** Takes place at the start of each sprint. In sprint planning meetings, scrum teams decide the business 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 inspect their progress and make adjustments to their plan to achieve their sprint goal and coordinate removal of impediments with the scrum master.
- » **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 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 product development. 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 as 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 two of the five days required for CSD; the remaining three days are a technical skills course focusing on extreme programming practices such as test-driven development, continuous integration, coding standards, simple design, and refactoring. You learn more about these practices in the next section. See <http://scrumalliance.org> for more about CSD certification.
- **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 product development, and passing an exam on your knowledge of agile fundamentals. See <http://pmi.org>.

Advanced and professional certifications are also available for all three scrum roles as well as agile leadership. See <http://scrumalliance.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. Ron Jeffries continues to update the XP practices on his website at <http://ronjeffries.com>.

The focus of extreme programming is customer satisfaction. XP teams achieve high customer satisfaction by working collaboratively with customers to develop the functionality the customer needs, 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 development effort now consists of people in customer, management, technical, and product 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 during development, not at the end.** 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.** With XP development, 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 will 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 5-1 lists key XP practices, most of which are commonsense practices and many of which are reflected in agile principles.

TABLE 5-1 Key Practices of Extreme Programming

XP Practice	Underpinning Assumption
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.
Planning game	All members of the team should participate in planning. No disconnect exists between business and technical people.
Customer tests	As part of presenting each desired feature, the XP customer defines one or more automated acceptance tests to show that the feature is working. The system always improves and never backslides. Automation is important because in the press for time, manual tests get skipped.
Small releases	Release value to the customer as often as possible. Some organizations release multiple times daily. Avoid building up large stores of unreleased code requiring extensive risky regression and integration efforts. Get feedback from your customer as early and as often as possible.
Simple design	The simpler the design, the lower the cost to change the software code.
Pair programming	Two people work together on a programming task. One person is strategic (the driver) and the other person is tactical (the navigator). They explain their approach to each other. They take turns as driver and navigator. No piece of code is understood by only one person. Defects can be more easily found and fixed before the code is merged and integrated with the system.
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.
Design improvement (refactoring)	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.

XP Practice	Underpinning Assumption
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. XP teams promote multiple builds each day!
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.
Coding standard	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.
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.
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.



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.

If you're not doing software product development, you can probably replace XP with a set of technical practices specific to your industry that would build in quality throughout product development.

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 5-2 shows a few more of the similarities among the three approaches.

TABLE 5-2

Similarities between Lean, Scrum, and Extreme Programming

Lean	Scrum	Extreme Programming
Engaging everyone	Cross-functional development team	Entire team Collective ownership
Optimizing the whole	Product increment	Test-driven development Continuous integration
Delivering fast	Sprints of four weeks or less	Small release

In addition to more extensive agile frameworks and practices, scrum also accommodates a variety of accouterments that consistently increase success with agile product development. 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 (an engaging elevator pitch — a clear inspirational statement of direction for reaching the outer boundary of the product)
- » Product roadmap (a representation of the features required to achieve the product vision)
- » Velocity (a tool—*not a metric*—for scrum teams to understand historical 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)

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

Chapter 6

Agile Environments in Action

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.

Or perhaps every one of the people we just listed is in a different physical location, maybe even in a different time zone, with no common physical workspace, as nearly all of us experienced in 2020 because of the stay-at-home orders related to COVID-19.

In addition, 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 product.

Although this description might not describe your particular situation, you most likely see parts of it in any given corporate setting. In contrast, scrum teams develop in short, focused iterative cycles, relying on timely feedback from team members and stakeholders. 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

Scrum teams flourish when scrum team members work closely together in an environment that supports continuous and close collaboration. As noted in other chapters, the development team members are central to success. Creating the right environment for them to operate in goes a long way toward supporting their success.

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, taking advantage of the fullness of both verbal and nonverbal communication
- » Standing up — rather than sitting — as a group for the daily scrum meeting (to keep 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

One of the benefits of collocation you don't recognize until it's gone is osmotic communication. When people work in the same physical environment, they hear what's going on around them, even if they're not paying close attention. They can join a conversation happening within earshot, contributing something that might have been missed. In addition, they can sense tension or relief as challenges are addressed by other members of the team. Absorbing information that's happening around you contributes to better informed and empowered team members.

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. And when the question is complex, a face-to-face conversation, with all the synergy it creates, is much more effective and efficient than any form of electronic communication. (See Principle 6.)

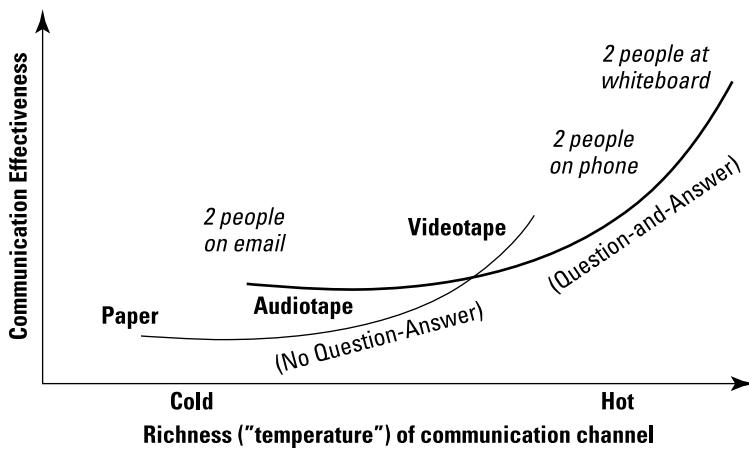


TECHNICAL STUFF

This improved communication effectiveness is due to *communication fidelity* — the degree of accuracy between the meaning intended and the meaning interpreted. Albert Mehrabian, PhD, 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. Keep this in mind during your next conference call to discuss the design nuances of a system that doesn't yet exist.

Alistair Cockburn, one of the Agile Manifesto signatories, created the graph in Figure 6-1. 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 6-1:
Better communication through collocation.



REMEMBER

Face-to-face communication means we are physically face-to-face as we communicate. Although technology today supports bringing geographically dispersed people together more effectively than ever before, technology cannot replicate the social effects of people working in the same physical location on effective and real-time collaboration.

Scrum teams are most effective when they are physically collocated. However, that doesn't mean agile frameworks such as scrum will not work for a dislocated team. In fact, for the success of a dislocated team, a framework such as scrum is

even more important due to its clarity of roles, transparency, and tight empirical feedback loop. You learn more about scrum roles, artifacts, and events in Chapters 7, 10, 11, and 12.

In the following sections, we describe the ideal situation for enabling scrum team communication, realizing that opportunities to achieve the desired benefits of collocation exist even when collocation isn't possible.

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 *team room* or a *scrum room*. The scrum team members create the setup they need in this 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, be creative and ask for a group of empty cubes and configure them in a way that works for your team, even if that means removing panels. Create a space that you can treat as your team room.



REMEMBER

The right space allows the scrum team to be fully immersed in solving problems and crafting solutions. Visualizing ideas and work-in-progress brings about shared understanding among the entire team. Unfettered access to other team members is also key for effective and real-time collaboration. Create a space that enables these things.

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 optimal conditions. Resources will vary, but at a minimum, they should include whiteboards, bulletin boards, markers, pushpins, and sticky notes. You'll be surprised at how quickly the efficiency gains more than pay for the investment.

For example, when one client company allocated a dedicated team room and made a \$6,000 investment in multiple monitors for developers, they increased productivity and saved almost two months and \$60,000 over the life of development. That's a pretty good return on a simple investment. We show you how to quantify these savings in Chapter 15.

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, let me quickly respond to this text message.

Okay, I'm back. The good news is that a scrum 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 6-1 is a list of common distractions and do's and don'ts for dealing with distractions.



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.

TABLE 6-1 Common Distractions

Distraction	Do	Don't
Multiple objectives	Do make sure that the development team is dedicated 100 percent to a single product objective at a time.	Don't fragment the development team between multiple objectives, 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 terms of lost productivity.
Over-supervising	Do let development team members decide among themselves how to accomplish the work to be done 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 sprint review meeting provides ample opportunity to assess progress.
Outside influences	Do redirect any distractors. If a new idea outside the sprint goal surfaces, challenge the product owner to add the item to the product backlog rather than put the accomplishment of the sprint goal at risk.	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. Assigning even a seemingly quick task can throw off work for the 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.

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.** Scrum teams stand throughout a meeting to discourage it from running longer than 15 minutes. They physically gather around the team's task board.
- » **Ask the product owner questions.** Also, make sure the product owner is involved in discussions about product features so he or she can provide clarity when necessary. The conversation shouldn't end when planning ends. In other words, make sure the product owner is accessible at any time and as close by as other team members. With movable desks and chairs, the scrum team, which includes the product owner, can reconfigure their space for better access to each other. Being mobile enables freer collaboration and more freedom overall.
- » **Speak with your team members.** If you have questions about features, progress, or integrating, talk, don't email back and forth with team members. 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
- » A clear definition of what it means to be shippable
- » 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 team's definition of done). See Chapter 11 to learn more about making team artifacts more transparent.



TIP

Our 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 are fantastic at facilitating communication.

- » Lots of colorful pens.
- » A sprint-specific task or kanban board (described in Chapters 5 and 11) for tracking progress tactiley.

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 product development or the scrum team.

Basically, you move sticky notes or cards around the board to show the status (see Figure 6-2). Everyone knows how to read the board and how to act on what it shows. In Chapter 11, you find out the details of what to put on the boards.



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. Pageantry is the enemy of agility.

RELEASE GOAL:	SPRINT GOAL:		
RELEASE DATE:	SPRINT REVIEW:		
TO DO	IN PROGRESS	ACCEPT	DONE
			<div style="text-align: center;"> US Task Task Task </div>
		US 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 6-2:
A scrum task board on a wall or whiteboard.

High-Tech Communicating

Although collocation almost universally improves effectiveness, many scrum teams can't be collocated. Some development efforts have teammates scattered across multiple offices; others have off-shore 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. Although high-tech tools are available to simulate a collocated environment, everyone's contributions are required to be effective. If one team member is remote, everyone on the team may need to be on camera to help the remote team member participate and be engaged.

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 the software display looked like a literal kanban board on the wall and was used in the same way. If the software tool had required filtering or scrolling to see all information on the kanban board, it would have been a significant downgrade for the teams, because what made their physical kanban board so effective was being able to see the board updated real-time in its entirety.

When determining which types of high-tech communication tools to support, don't choose tools that degrade direct and real-time conversation or introduce unnecessary complexity:



REMEMBER

- » **Videoconferencing:** Videoconferencing tools, such Zoom, Teams, and Hangouts, 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.

Decisions aren't made during meetings — they're communicated in meetings. Decisions are made in less formal settings, such as during a hallway conversation, over lunch, at the water cooler, or in an impromptu discussion in someone's office. Unless you're face-to-face, replicating these types of interactions is difficult. The closest you might get is using something like a telepresence robot (a remote-controlled stand on wheels with a tablet on top that displays the remote person's face). The robot can move around an office space so that the single remote person can have hallway and water cooler conversations, almost as if he or she were physically there. Probably not a realistic option, but it shows that you can't beat face-to-face interactions.

- » **Persistent chat:** Although instant messaging doesn't convey nonverbal communication, it is real time, accessible, and easy to use. Current examples include Slack, HipChat, and Teams. Several people can also share a session and share files. Persistent chat can be useful for conveying information but not for resolving issues. Have face-to-face conversations for resolving issues.
- » **Web-based desktop sharing:** Especially for the development team, sharing your desktop allows you to visually highlight issues and updates in real time. Seeing the problem is always better than just talking about it hypothetically over the phone. Most videoconferencing tools include this capability.

» **Collaboration tools:** These tools allow you to do everything from sharing simple documentation so that everyone has the latest information to using a virtual whiteboard for brainstorming. Some examples include tools such as Google Drive, Miro, Mural, and Jamboard.

Technology is constantly evolving. We can't wait to see what tools are available by the time we release the next edition of this book.



TIP

Using online collaboration tools (such as those just listed) allows you to get out of the status-reporting business and focus your efforts on doing the real work of creating value for your customers. Use these tools to make the artifacts you're already using (such as your sprint backlog) available for stakeholders to peruse on demand. When managers request status updates, you can simply direct them to the collaboration site to pull the information they need. 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 that comes from conversations.

Choosing Tools

As noted throughout the chapter, low-tech tools are best suited for agile development, especially initially, while the scrum team becomes accustomed to a more agile way of working and collaborating. Our advice on choosing a tool is less about which specific tool is best and more about whether the tool enables or impedes delivering value to your customer.

We like to start our teams off with low-fidelity tools such as whiteboards, flip-charts, sticky notes, and markers. Later, if they feel they need to invest in more sophisticated technology, we encourage them to find a tool that supports the way they've found works best for them, rather than find a new tool that dictates how they do their work.

In this section, we describe 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.

Good litmus tests for tools include:

- » As a development team member, can I update my task status in a minute or less per day?
- » Does the tool help us do our work, or has administering it become our work?
- » Is the tool impeding or improving transparency?
- » Does it promote or hinder important face-to-face conversations?
- » Does the cost of the tool’s administration justify the need?
- » Does the tool enable leadership interaction consistent with agile values and principles?



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 product issue.

Tools that encourage the success of forced team dislocation

During the COVID-19 pandemic of 2020, governments worldwide issued stay-at-home orders that forced a significant majority of people throughout the world to work remotely — dislocated — for months. Countless organizations and industries pivoted product delivery, collaboration, and even business models. Disruption spanned almost every industry.

Our business, Platinum Edge, was no exception. Before the pandemic, the scrum certification programs we delivered, such as Certified ScrumMaster (CSM) and Certified Scrum Product Owner (CSPO), were required by Scrum Alliance to be taught only in-person, physically in a classroom setting. This requirement was temporarily lifted, enabling us to deliver much needed training and experience to people live and online. The experience was powered by digital tools that brought students and our Certified Scrum Trainers together for a valuable learning experience.

We quickly adapted to this challenge and found ways to enable effective collaboration and learning in a virtual environment. We used videoconferencing with breakout rooms for small-group collaboration. Figure 6-3 shows a student's computer screen of the virtual classroom setting.



FIGURE 6-3:
Virtual classroom setting.

We also shared visual content (prepared slides as well as real-time drawing) to explain concepts, and enabled small groups to practice and apply concepts on a virtual working canvas that simulated many of the tools mentioned in this chapter (see Figure 6-4). Virtual flipcharts, whiteboards, and sticky notes provided a near-tactile experience of moving, removing, and creating items to share.

The experience wasn't the same as being physically face-to-face, and given the choice, we and our students would choose in-person interactivity. But it was sufficient for effective learning, "better than expected" to quote quite a few students.

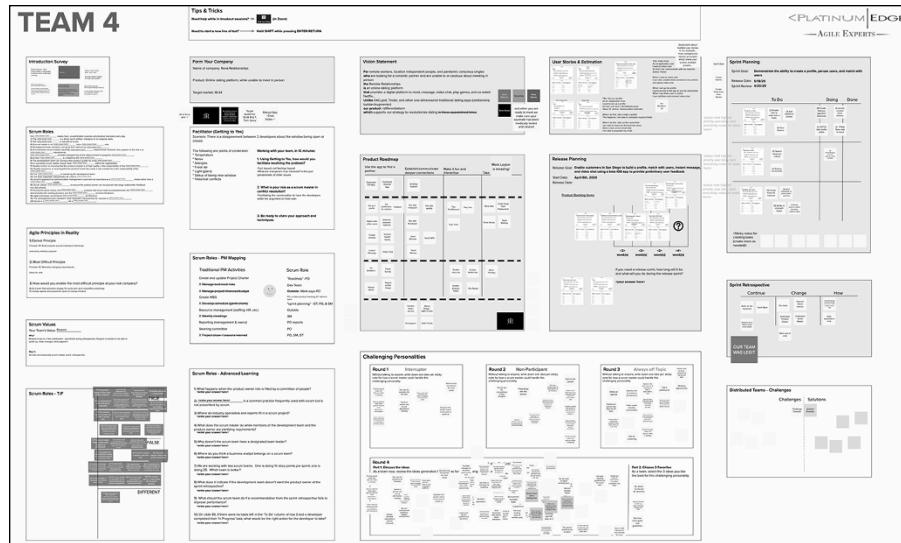


FIGURE 6-4:
Virtual collaboration board.

Using these tools, the students practiced all the same activities and experienced working as a scrum team just as they did in the real world. As a side benefit, the students were able to retain the virtual board after the class for their continued collaboration.

Although collocated teams build better products faster, many students were able to successfully navigate their forced virtual working conditions by using high-tech tools. Other organizations who invested in tools to help their employees work virtually continued operations even during a worldwide pandemic.

Organizational and compatibility constraints

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.

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

IN THIS CHAPTER

- » Clarifying agile roles
- » Embracing agile values in your organization
- » Transforming your team's philosophy
- » Sharpening important skills

Chapter 7

Agile Behaviors in Action

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 a product development team and see how you can change a team's values and philosophy about product development. Finally, we discuss some ways for a team to hone key skills for agile success.

Establishing Agile Roles

In Chapter 5, we describe scrum, the most popular agile framework 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 product development:

- » Stakeholders
- » Agile mentor

The scrum team together with the stakeholders make up the *product 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 7-1 shows how these roles and teams fit together. This section discusses these roles in detail.

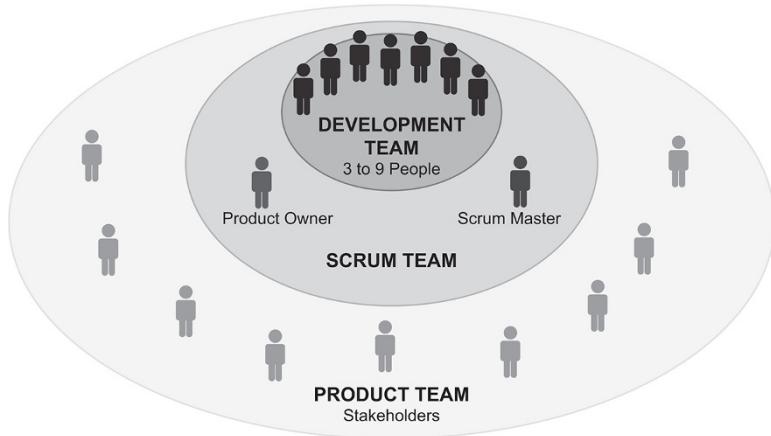


FIGURE 7-1:
The product team, scrum team, and development team.

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 (competing priorities), 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.

With agile product development, the product owner will

- » Develop strategy and direction for the product and set long- and short-term goals.
- » Maximize the value of the product resulting from the work of the development team.
- » Provide or have access to product expertise.
- » Understand and facilitate discussions about the customer's and other business stakeholders' needs with 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.



REMEMBER

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.

Figure 7-2 shows how a product owner works with stakeholders, customers or users, and their scrum team. These relationships are critical for communicating across the organization and for receiving product feedback.

Table 7-1 outlines the responsibilities and matching characteristics of a product owner.

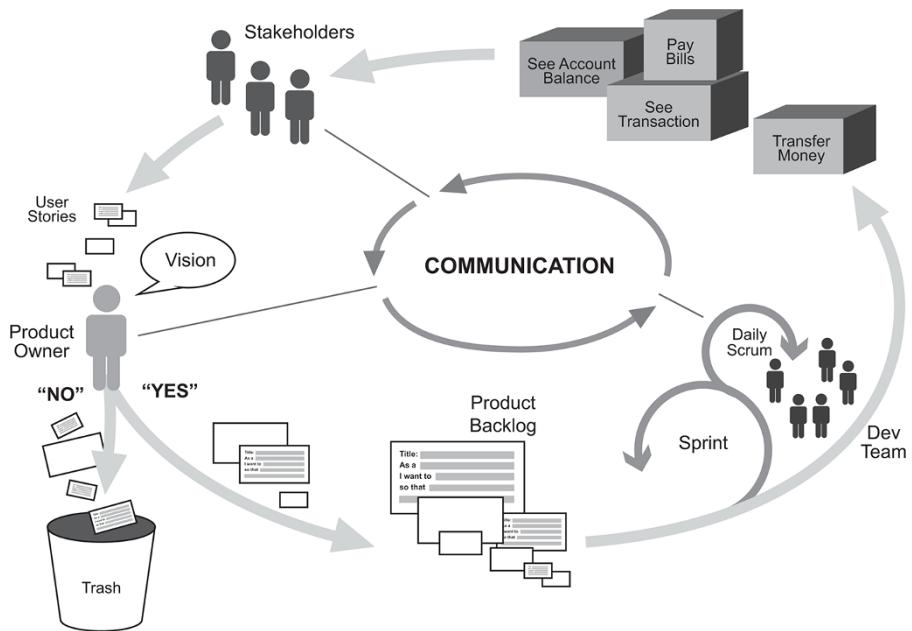


FIGURE 7-2:
Product owner communication cycle.

Characteristics of a Good Product Owner

Responsibility	A Good Product Owner ...
Supplies product 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 effectiveness 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) and has the courage to say "no"

Responsibility	A Good Product Owner . . .
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 and technical risks Collaborates well with developers Adeptly describes product features
Accepts or rejects work	Understands acceptance criteria of requirements and ensures that completed functionality 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 development. Although the sponsor funds and owns the budget, the product owner manages how the budget is spent.

Product discovery

Product discovery is a key responsibility of product owners, who are often found in the field performing customer interviews, striving to better understand challenges and problems. They meet with stakeholders and host workshops to gather ideas for improving ROI and product value. They discover what the product needs to become.



WARNING

Product discovery is not a relabeling of a waterfall's big, up-front planning phase. Product discovery is ongoing, through continuous inspection and adaptation of the customer's needs.

Gathering data is another activity performed by product owners. They watch product usage trends, looking for problematic areas. They dive into customer service issues to understand patterns and opportunities for improving the customer's experience. With the help of talented experts in user interface (UI), user experience (UX), engineering, and other subject matter, they continually look for opportunities to improve product design. See Chapter 11 to learn other product discovery activities performed by a product owner.

Product ownership is a full-time, fully dedicated role because they not only perform product discovery but also enable product development.

Product development

As the product is developed, the product owner is key for helping the team achieve sprint and release goals. Throughout development, many questions will arise needing clarification. The product owner attempts to answer these questions.

They do the necessary preparations before the release and sprint planning, product backlog refinement activities, and sprint reviews and retrospectives (discussed in Chapters 10 and 12). They watch the team's task board. If a task is taking longer than expected or blocked, they work with the scrum master and development team to find opportunities where they can help. See Chapter 11 to better understand how the product owner works throughout the sprint to support product development.

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

Development team member

Development team members are the people who create the product. In software development, programmers, testers, UI designers, writers, data engineers, UX designers, 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.

With agile product development, the development team is

- » **Directly accountable for creating deliverables — the product features and functionality.**
- » **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, including skills needed to automate the deployment of their product increments. High-performing development teams have all the skills needed to create and release product increments to the customers and users.



REMEMBER

» **Multi-skilled.** Development teams aren't just cross-functional as a whole; development team members are also versatile — they're not tied to a single skill set. They have existing skills to immediately contribute at the beginning of development, but they are also willing to learn new skills and to teach what they know to other development team members.

Every developer should be able to do more than one skill. And every skill should be able to be done by more than one person. Specialized skill developers ("I do only one thing") are great until they're not. Scrum teams need access to all skills every day, not just at certain phases in development as in waterfall projects. If the one developer who knows how to run tests calls in sick even one day, the whole team is unable to get to done for that day. You need at least one other person on the team who, in a pinch, can step in to keep the team moving forward.

» **Ideally dedicated to one product objective for the duration of development.**

» **Ideally collocated.** The team (including the product owner and scrum master) should be working together in the same area of the same office, preferably in a team room.

What makes a good development team member? Versatility. You want developers who are intellectually curious — who look for ways to contribute to the sprint goal a little more today than the day before. Some development team members aren't fully versatile at first. Ideally, during the team's work, developers with one skill take the opportunity to gain exposure across the stack of skills as they shadow and pair with other developers, becoming more T-, Pi-, or M-shaped in their skillset, as shown in Figure 7-3.

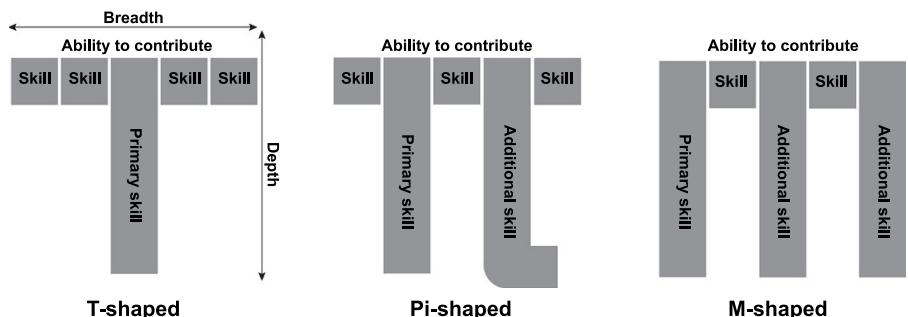


FIGURE 7-3:
Development
team
member skill
development.

High-performing scrum teams have developers whose skill set is Pi (π) or M shaped. In other words, in addition to their primary skill and their broad exposure to all the skills needed by the team, they are proficient in one more skill

(Pi-shaped) or two more skills (M-shaped). Development teams with Pi- and M-shaped skilled team members typically have a higher velocity because they eliminate single points of failure.

Take a look at the team responsibilities and matching characteristics in Table 7-2.

TABLE 7-2 Characteristics of a Good Development Team Member

Responsibility	A Good Development Team Member . . .
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 *facilitator* or *team coach* in non-scrum 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 16.

With agile product development, the scrum master will

- » Act as a process coach and agile champion, helping the team and the organization follow scrum values and practices.
- » Help remove impediments — both reactively and proactively — and shield the development team from external interferences.
- » Work with the product owner to foster close cooperation between stakeholders and the development 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 teams 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 challenging the status quo to remove roadblocks and prevent distractions to the development team's work. A scrum master who is good at these tasks is priceless to product development and to the organization. If a development team has seven people, the effect of a good scrum master is times seven.

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. He or she knows the right questions to ask to guide the team to higher performance through introspection and retrospection. The scrum master also works 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 7–3.



TIP

Clout is not the same as authority. Organizations need to empower their scrum masters so they can influence change in the team and organization without formal authority over others. 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 organizational clout.

TABLE 7-3 Characteristics of a Good Scrum Master

Responsibility	A Good Scrum Master . . .
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 product objectives and the current sprint
Fosters close cooperation between external stakeholders and the scrum team	Looks at the needs of the product team 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

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

As we mention earlier in the chapter, the scrum team plus stakeholders make up the product 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 product.

Stakeholders

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

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*.

On the count of three, each person holds up a number of fingers, reflecting their degree of comfort with a proposed path forward as a way to deal with the issue 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. After all group members show at least three fingers — they don't need to love the idea, but they can support it — you have consensus and can move forward. 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 (either way is fine). Some people refer to this as a Roman vote. It's quicker than a fist of five, and is great for answering yes-or-no questions.

With agile product development, stakeholders

- » Include the customer
- » May include technical people whose subject-matter expertise can support the development team to do its work
- » May include the legal department, account managers, salespeople, marketing experts, and customer service representatives who can affect the product
- » May include product or market 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 products and organizations. Almost all product teams have stakeholders outside the scrum team.

Some product teams also have agile mentors, especially 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 principles, practices, and techniques, and can share that experience with a team. The agile mentor can provide valuable feedback and advice to new teams and to teams wanting to perform at a higher level.

With agile product development, 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 diversity of experience in implementing agile techniques for products of varying contexts

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 are

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

The following sections provide details about each of these values.

Commitment

Commitment implies engagement and involvement. With agile product development, 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. There is no need for managers to hold scrum teams accountable for every specific task they identify could be done to achieve their business goals. Tasks may change, but business goals are what matter and drive outcomes. Scrum teams hold each other accountable because they are purpose-driven or outcome-driven. With strategic stability, they maintain tactical flexibility.

Commitment requires a conscious effort. Consider the following points:

- » Scrum teams must be realistic when making commitments, especially for short sprints. Aim high, but not unrealistically high.
- » 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 product. 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 15, you read about how to use performance to accurately determine schedules and budgets.

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 up front to create a distraction-free environment that helps your team succeed.

Openness

Secrets have no place on a scrum team. If the team is responsible for the result of the product, it only makes sense that the team has all the facts at its disposal. Information is power, and ensuring that everyone — both the scrum team and stakeholders — have access to the information necessary to make the right decisions requires a willingness to be transparent. They agree to be transparent with not only the progress of work but also the challenges with performing the work. To leverage the power of openness, you can

- » **Ensure that everyone on the team has access to the same information.** Everything from the product vision down to the smallest detail about the status of tasks needs to be in the public domain as far as the product 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 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 product'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. Scrum team members respect each other as capable, independent people. 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.

Courage

The last scrum value is courage. We list it last because it requires courage to live the other four scrum values. In other words, it takes courage to commit to goals. It takes courage to focus. It takes courage to be open, and it takes courage to show and expect respect. Bottom line, it takes courage to do scrum — not just at first, but always.

Embracing agile techniques is a change for many organizations. Successfully making changes requires courage in the face of resistance. It takes courage to do the right thing and to work on tough problems. 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.
- » **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 because they often created the old rules. You're challenging those rules. Respectfully remind them 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 scrum 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.

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 product development in a whole new way to achieve their commitments.

To be successful scrum teams should embrace the following attributes:

- » **Dedicated team:** Each scrum team member works only on the product objectives decided by the scrum team, and not on other teams or products. Product development may finish on one product, and then new development may start on a new product, but the team stays the same long-term.
- » **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.



TIP

- » **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.

We've found that development teams of four to six people are the right size. The lines of communication to self-organize around the work are not too complex, adequate skill coverage exists, and the cost is manageable.

- » **Ownership:** Take initiative for work and responsibility for results. Have pride in your craftsmanship.

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* — people are not inanimate objects) 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. Work in progress has no value; completed work does.

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 product objectives 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 product objective at a time.

The following results occur when you dedicate scrum teams to work on only one objective 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 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 competing priorities.
- » **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 out on the team with one skill, but learn to perform many different jobs throughout as they help create the product.

Cross-functional development teams collectively possess all the skills required to take product requirements from idea to delivered value. But cross-functional teams aren't enough. Cross-functional *individuals* make development teams more efficient.

For example, suppose a daily scrum meeting uncovers testing as the highest priority task to complete the requirement that day. A programmer or a designer who also has some skill at testing steps in to help and finish the task more quickly, or possibly in the absence of another development team member who called in sick that day. When the development team is cross-functional, it can *swarm* (a

technique for limiting work in progress) on one product requirement at a time, 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 a scrum 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 during each sprint. Techniques such as *mob programming* — where entire teams work together to code one item — can help you learn new skills quickly and increase overall product quality. A more detailed explanation of mob programming is discussed in Chapter 11.
- » **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 the next highest priority piece of functionality today rather than starting on something of lower priority. That development team member learns and improves, and the workflow continues to be balanced. Cross-functional teams enable the next available person to pull the next task needing to be completed, whatever that might be (rather than having someone push work to the person).

One big payback of cross-functionality is that the development team completes work quickly. Post-sprint review afternoons are often celebration time. Go to the movies together, play video games, head to the beach or the bowling alley, or 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 development teams do not have a project manager to tell them 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 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. Learn more about using effective mediums of communication in Chapter 6.

- » **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 scrum team. Development teams take input from stakeholders but own their final solution.



REMEMBER

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

- » **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, but 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 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. If a requirement isn't completed but there are no new tasks to start, before beginning a new requirement (and increasing the team's work-in-progress), ask, "Is there someone you can help?" or "Is there something you can learn by shadowing?" 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 and pride in their craftsmanship.

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 product is successful. To succeed with self-management, development teams

- » **Allow situational leadership to ebb and flow.** With agile product development, 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, not title.

- » **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. In Chapter 11, 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: 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 it has 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 product'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. (Each additional team member increases team communication channels geometrically — that's not exponentially, but it's close — as shown in Figure 7-4.)
- » Maintains the team in a single unit.
- » Promotes joint ownership, cross-functionality, and face-to-face communication.

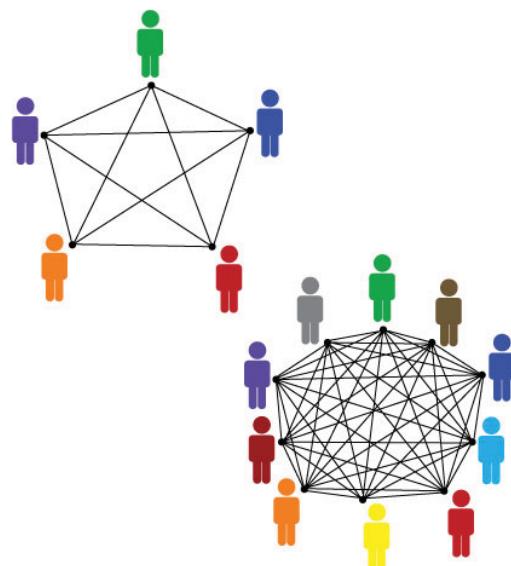


FIGURE 7-4:
Team communication complexity is a function of team size.

When you have a small development team, a similarly limited and focused 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 risk.

When you have a large product and a correspondingly large development team, split the work between multiple scrum teams. For more on scaling — or better yet de-scaling — across the enterprise, see Chapter 19.

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 products and results. Even seasoned development team members may need to adjust their behavior to get used to making decisions.

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.** With agile product development, accomplishments and failures alike belong to the 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.

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.

IN THIS CHAPTER

- » Understanding why permanent teams are needed
- » Figuring out what motivates people
- » Seeing why and how permanent teams continuously improve knowledge and capability

Chapter 8

The Permanent Team

Describing a product development team as a *permanent team* may seem a bit extreme. How can a team working in today's ever-changing business environment become permanent? Product development teams who work together consistently on long-lived products become more effective with time. Certainly, team adjustments may occur so that team members can pursue career growth or other business opportunities, but each change in the team's composition causes the team to step back and relearn. For this reason, teams should be as long-lived, stable, and enduring as possible for sustained quality product development.

Enabling Long-Lived Product Development Teams

Today's products are required to meet a multitude of both near-term and long-term needs. With traditional project management, the horizon for realizing value from a product development project spans months to a year or even multiple years. Return on product-focused investments, however, can start much earlier and last much longer. It's not uncommon for a product to last and evolve effectively for six or ten or more years. New ideas or needs are continually identified through maintenance and enhancement requests. Stable, enduring, and long-lived

product development teams are best suited for building long-lived, valuable products.

Instead of being a cost center, high-performing teams can become a revenue and cost-saving center. They become valuable organizational assets capable of tackling difficult problems.

At an individual level, long-lived teams become almost like a family. Family-sized. Vulnerable and honest with each other. Accomplishing challenges or hurdles placed in its path. They may even eat or socialize together, and they definitely learn and work together day in and day out. Being part of a great team can be one of the most rewarding experiences of a person's career.

Peter Senge, in his book *The Fifth Discipline: The Art and Practice of the Learning Organization* (Doubleday), wrote: "When you ask people what it is like being part of a great team, what is most striking is the meaningfulness of the experience. People talk about being part of something larger than themselves, of being connected, of being generative. It becomes quite clear that, for many, their experiences as part of truly great teams stand out as singular periods of life lived to the fullest. Some spend the rest of their lives looking for ways to recapture that spirit."

Agile principles and values were defined to help teams work together. Keep in mind the following key principles:

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.

Leveraging long-term knowledge and capability

Product development teams use empirical process controls to inspect and adapt and learn. They learn about the customer, users, product, architecture, and sponsors. Every day they learn better techniques for improving their product and working more effectively with each other. Because the teams are ideally collocated, they know their teammates' personal lives, work aspirations, dreams, and goals. With time, they develop a valuable shared memory that they use as a reference for all their work.



WARNING

Handoffs are one of the “Seven Wastes of Software Development” mentioned by Mary and Tom Poppendieck in their book *Implementing Lean Software Development: From Concept to Cash* (Addison-Wesley Professional). Half of a team’s knowledge is lost during a hand-off, and the situation compounds — if Bob tells Jim who tells Sue, then 75 percent of the knowledge is lost. Asking someone else to pick up where you left off requires costly context switching, repeated research, and a re-understanding of the product work. Swapping team members into and out of the team and asking them to hand off their work can be costly.

Product development teams become incubators for learning. Capability is developed as the team works together. When team members learn something new that will benefit their team, they are excited and willingly share it. The team creates a safe environment for experimentation, failure, and vulnerability.

Teammates want each other to be successful. Everyone on the team knows that the more skilled they become, the more effective they’ll be at creating better products faster. Sprint review after sprint review and sprint retrospective after sprint retrospective uncover new understanding that incrementally move the team forward. (Find out more about sprint reviews and sprint retrospectives in Chapter 12.)

From a product perspective, over time product development teams gain a deep understanding about the product and its architecture. They understand intimately how the product functions, what to test for, areas to avoid, product documentation, and much more. Change is easily implemented because they understand every part of the product that could be affected by the change. Emergent architecture comes to life because the team builds an architecture that is barely sufficient for the current need and evolves when needed.

Staffing product development teams for the long-term maximizes the benefit for the product, the organization, the team, and the individuals on the team.

Navigating Tuckman’s phases to performance

When teams learn to work with one another, the process is known as *team development*. Bruce Tuckman, an educational psychologist, outlined the following five phases that most teams follow:

» **Forming:** During this phase, the team orients itself and gets acquainted. In this period of uncertainty, team members seek to understand common expectations, personal fit within the team, and how they’ll benefit from interacting with their teammates. Team interactions are social as team members get to know one another.

- » **Storming:** This phase can be the most visibly difficult for the team. After team members become acquainted, the realization sets in that they must work with each other in accomplishing a goal. Personality types clash, interpersonal conflicts arise, frustrations increase, and competition heightens. This phase, if not carefully facilitated by a scrum master, can bog down the team and result in long-term problems. Team performance is often at its lowest during storming.
- » **Norming:** If the team successfully emerges from storming, a sense of unity emerges. Roles become clearer, interpersonal differences are resolved, and performance begins to improve. If conflict goes unresolved, the team may slide back into storming.
- » **Performing:** During this phase, the team begins to hit its stride. Alignment on goals is clear and each person begins to understand how they can best help their teammates. Problems and conflicts may arise, but they are dealt with constructively. The team experiences an openness and transparency with one another that leads to improved performance.
- » **Adjourning:** At some point far in the future (we hope), the product or team may reach its conclusion and end. If a new product is started for an existing team, however, the team will experience a bit of adjourning as it transitions from the old product to the new product. The team's journey may navigate again through the previous four stages.

No team escapes navigating these phases, and no team can skip a phase because each is crucial for the team's development. Many teams go through multiple stages more than once. Unfortunately, many teams wander the storming phase longer than they want. Yet the storming phase, like the refiner's fire, makes reaching the performing phase as a team that much healthier. Storming builds team character, alignment, and strength.



TIP

Address the storming phase head on. Each interpersonal conflict and poor behavior must be addressed. Scrum masters lead the team in inspecting its working agreement (discussed in the “Creating a working agreement” section) to build consensus, improved behavior, and team alignment.

Disrupting a performing team with staff changes sends the entire team, at least to a certain degree, back to the forming phase as each person once again learns how the change effects where they fit. High-performing teams should be leveraged rather than disrupted to build new teams. Agile organizations encourage teams to pull from an organizational backlog of value-driven opportunities rather than push people to staff traditional projects.



WARNING

Team member thrashing (multitasking) is costly. Effective multitasking is a myth. High-performing teams focus on one objective, do it well, and then attack the next most important objective. Thrashing is a failure of prioritization. If you want to improve performance, change the structures in place for prioritization. See Chapters 5 and 7 to learn more about the evils of thrashing.

The journey to become a high-performing team is not easy. Each of Tuckman's phases is filled with challenges, learning opportunities, and growth. When a team performs together well, keep it together as long as you possibly can.

In their famous 1986 article “The New New Product Development Game,” which started the scrum analogy, Hirotaka Takeuchi and Ikujiro Nonaka revealed that high-performing teams share three attributes: They are autonomous, self-transcendent, and cross-fertilized. A high-performing team discovers and delivers outcomes that have significant effects on the customer:

- » **Autonomous** in that they are able to set their own direction and act independently. They are self-organizing and self-managing.
- » **Self-transcendent** in that the team pursues pushing the limit, continually establishing and then elevating its own goals.
- » **Cross-fertilized** in that they are a diverse, multi-skilled team willing to share its knowledge and expertise.

These three factors above all else lead to high performance.

Focusing on fundamentals

Sports coaches often say, “Practice the fundamentals.” The same is true for product development teams. High-performing teams became high performing because of their mastery of the fundamentals. This mastery is gained through experience. Understanding the fundamentals allows the team to continually improve its execution or team development. The migration of an team’s journey to improved performance is often referred to as *Shu Ha Ri*.

Shu Ha Ri, a Japanese marital art concept in Aikido, is used to describe the stages of learning to mastery. In Chapter 20, we discuss *Shu Ha Ri* in the concept of maturing and solidifying organizational change movements. In this chapter, it’s helpful to understand the importance of achieving *Shu* to get the fundamentals right first.

- » **Shu (obey):** During this initial phase students follows the teachings of one master precisely. The concentration is on how to do the task. Multiple ways may be possible for completing the task, but the students focus on the one way taught by the master. Students commit the skill to memory and make it automatic.

- » ***Ha (detach)***: At this point, students begin to branch out. They start to learn the underlying principles and theory behind the technique. They begin to consider ideas from other masters and incorporate those ideas into their practice. Students learn more about the skill from successes and failures as they improvise.
- » ***Ri (transcend)***: During this phase students begin learning from their own practice rather than from other people. Students create their own approach and adapt what they've learned to their own circumstance. At the end of this phase, the skill comes naturally to these now-former students, who will know what works and what doesn't.

Shu Ha Ri focuses early stages of learning on steps to imitate, and then shifts to understanding principles and lastly to self-directed innovation. Solid understanding of the fundamentals make *Ha* and *Ri* possible.

High-performing teams use *Shu Ha Ri*. They learn and practice extensively the agile fundamentals over and over again. Sprint after sprint, their learning and performance improve, until they move into *Ha*, which brings a new set of learning for the team. *Ri* finally comes, but only after time and extensive effort have been invested in *Shu* and *Ha*, again reinforcing the importance of a long-lived team.

Creating a working agreement

Fundamental to a team's success is their *working agreement*, or expected norms for team behavior. Working agreements are guidelines defined by the team (not imposed on the team) as to how it agrees to work together to create a positive, productive development process. Many teams write out their agreements, sign them, and then post them on a wall for quick reference and constant reminders. A team will empower a team facilitator (often the scrum master) to help the team self-manage the agreement. Wise scrum masters help their teams to self-evaluate behavior against the agreed-to norms.

Working agreements benefit teams by helping them to

- » Develop a sense of shared responsibility
- » Increase awareness of each member's own behavior
- » Empower the facilitator to lead the group according to the agreements
- » Enhance the quality of the group process

Agreements work well when the agreement items are important to the team (meaning the team identifies the need and how to handle it), are limited in number, and are fully supported by each team member.

Examples of typical working agreement topics may include the following:

- » Meetings: Start and end times, meeting behaviors, and etiquette
- » Working together: Participation expectations, transparency methods, and consensus building

Team working agreements build behavior alignment. The best working agreements are defined by the team and for the team and are enforced by the team. Keep in mind that working agreements are different for every team. Just because one team struggles with a particular behavior doesn't mean every team's agreement needs to address that behavior. The best working agreements are brief — three to five items — making them easy to remember and self-enforce. The best working agreements should address only those items important to your team.

Enabling Autonomy, Mastery, and Purpose

Many organizations debate the best way to motivate people. Some question if money or other tangible rewards are motivating. Is it better to use the proverbial stick or carrot? Daniel H. Pink, in his best-selling 2009 book *Drive: The Surprising Truth About What Motivates Us* (Riverhead Books), reveals from his research that true motivation comes by giving team members autonomy, mastery, and purpose.

Autonomy

Autonomy is the freedom or independence to be creative in problem solving — to be the captain of your own destiny. High-performing teams are made up of people who are free to solve customer problems in the best way they see fit. Autonomy allows the team to become self-organizing and self-managing. High-performing teams can also behave autonomously, becoming empowered to do what is needed for their customers, sponsors, and stakeholders.

Mastery

Mastery is the internal drive the team demonstrates to excel in its trade. The opportunity to become curious and learn necessary skills, even multiple skills, is rewarding. Team members have pride in their craftsmanship and use their mastery to build quality into their products. After a skill or concept is learned, team members are quick to share what they've learned so that everyone on the team can improve. "Learn then teach" reinforces and supports the team's mastery.

Purpose

Purpose is the transcendent overarching objective for product development teams. It's the greater good for which they strive and give their full effort. The purpose is what brings them to the office each day feeling passion and determination for accomplishing great things. Studies, including a 2018 *Harvard Business Review* article written by Shawn Achor, Andrew Reece, Gabriella Rosen Kellerman, and Alexi Robichaux ("9 Out of 10 People Are Willing to Earn Less Money to Do More Meaningful Work"), show that people are willing to earn less money to do more meaningful work.

Teams and organizations who enable autonomy, mastery, and purpose are on the path toward improved team member motivation. As Principle 5 states, "Build projects around motivated individuals. Give them the environment and support they need, then trust them to get the job done."

TEAMMATES HELPING TEAMMATES

An extremely talented development team member was passionate about the quality of his team's product. At times, however, his passion became contentious as he berated his teammates for their lack of attention to detail and poor quality. Like a storm blowing in and leaving a wake of scattered debris, each occurrence lowered the entire team's energy and motivation. Discussions about this team member with HR were frequent.

To help his teammate, the scrum master called each morning during their morning commute to hold what they called a venting session. The scrum master asked questions, and then just listened while his teammate shared his concerns, fears, and frustrations. Never judging, but simply trying to be his friend, the scrum master was able to help his teammate reframe his perspective in preparation for the day of work.

The results from the daily ritual were amazing! The team member trusted the scrum master, having worked with him for an extended period, and was able to adjust his behavior. Teammates recognized the change and the entire team improved — it started reaching its sprint goals and more!

Both the team member and the scrum master look back on the experience with fondness, recognizing the deep foundation for their friendship, which has lasted longer than their employment with the organization. Enduring team relationships matter.

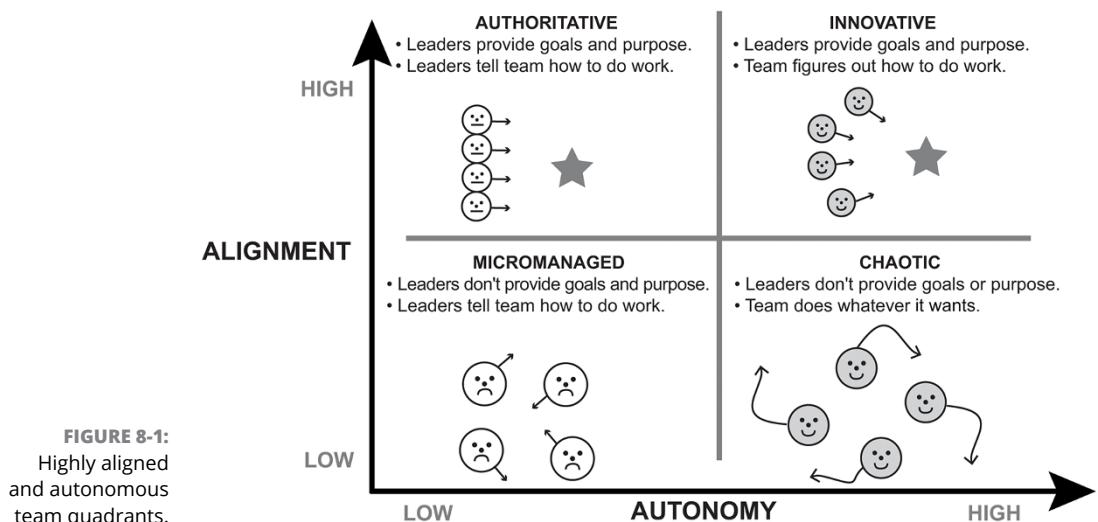
Highly aligned and highly autonomous teams

High-performing agile organizations have highly aligned and highly autonomous teams — almost like decomposing a large organizational ship into smaller speed boats, all pointing in the same direction.

Team autonomy helps the team to thrive. The team becomes empowered with the freedom and independence to do whatever is necessary to help the organization accomplish the aligned goal. Team autonomy helps each team leverage its uniqueness in solving the problems that only it can solve.

Leaders in agile organizations set strong visions for their teams but allow the teams to determine for themselves how to reach that vision.

Figure 8-1 describes the leaders' role in building alignment and autonomy across four quadrants from low to high. The maximum benefit is achieved when teams become both highly aligned and highly autonomous.



Building Team Knowledge and Capability

High-performing teams invest in education. They understand that learning is essential for building knowledge and capability. Although training in specific skills can be valuable, many teams find that becoming part of a community of practice or guild is also helpful.

A *community of practice* (*CoP*) is a group of people who share a common concern, set of problems, or interest in a topic and who come together to fulfill both individual and group goals. For example, product owners who want to learn how to improve their role participate in a community of practice with all other product owners, or people interested in learning more about product ownership participate to develop their skills. They discuss their challenges and keys for success, inspiring each other to experiment in their own situation. They collectively engage additional help, mentoring, and coaching from experts beyond their own expertise.

Communities of practice often focus on sharing best practices and creating knowledge to advance a domain of professional practice. Interaction on an ongoing basis is an important part of this. Many communities of practice rely on face-to-face meetings as well as web-based collaborative environments to communicate, connect, and conduct community activities.

Many communities of practice use the lean coffee approach to generating and discussing important topics, which is a simple way to ensure that the topics people care about most are given the majority of the community's time.



TECHNICAL
STUFF

Lean coffee is a simple meeting facilitation technique. Participants take a few minutes at the beginning to brainstorm ideas they'd like to discuss with the group. Once submitted, the ideas are organized into themes. Each person is given an opportunity to then cast five dot votes (dots on topic Post-it Notes with a marker). Each person can allocate their dot votes however they like, whether it's placing five on one topic or one on five separate topics. Topics receiving the most votes are prioritized first. Each topic is given a set amount of time for discussion, say 8 minutes. When the time expires, the group votes using thumbs up or down to add 8 more minutes, then 5, then 3, and then they either agree or disagree to stop and move to the next topic. Participants leave lean coffee discussions having covered the topics most valuable to them.

Communities of practice can be formed for all other team roles, interests, or organizational disciplines such as architecture, user experience, security, training, and customer support. The sky is the limit! Community members leave their community of practice discussions with pragmatic ideas for building team knowledge and capability.

By understanding the hard work, dedication, learning, and capability development required to become a high-performing team, it stands to reason that the team will maximize its contribution the longer it stays together. Customers who use long-term products benefit most from high-performing, long-lived, and enduring teams.



Agile Planning and Execution

IN THIS PART . . .

Follow the Roadmap to Value, from product 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.

IN THIS CHAPTER

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

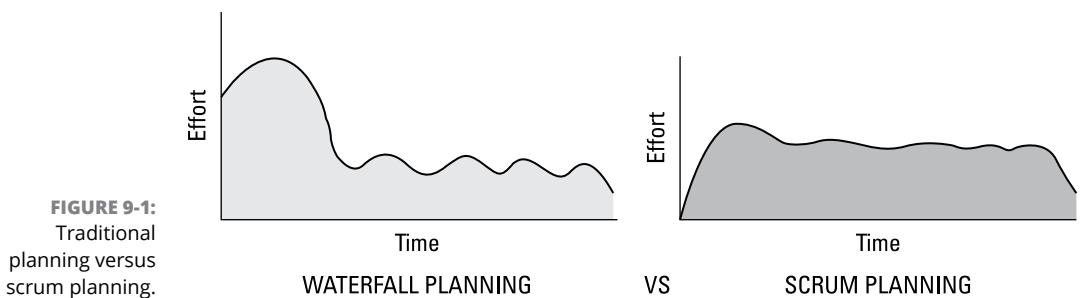
Chapter 9

Defining the Product Vision and Product Roadmap

To start, let's dispel a common myth. If you've heard that agile product development doesn't include planning, dismiss that thought right now. You will plan not only the overall product but also every release, every sprint, and every day. Planning is fundamental to agile product development 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 product development, in contrast, involves planning up front and throughout the entire product lifecycle. 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 success. Scrum teams plan as much as, if not more than, traditional project teams. However, agile planning is more evenly distributed throughout the product's life (see Figure 9-1) and is done by the entire team that will be working on the product.



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 developing a product feature — plans always change. Just-in-time planning allows you to accommodate real-world changes non-disruptively and to be well-informed as you plan specific tasks.

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

Agile Planning

Planning happens at a number of points. A great way to look at planning activities is with the Roadmap to Value. Figure 9-2 shows the roadmap as a whole.

The Roadmap to Value has seven stages:

- » In stage 1, the product owner identifies the *product vision*. The product vision is your product's destination or end goal. The product vision includes the outer boundary of what your product will be, how the product is different than 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. The product vision should be revisited 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 development. 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.

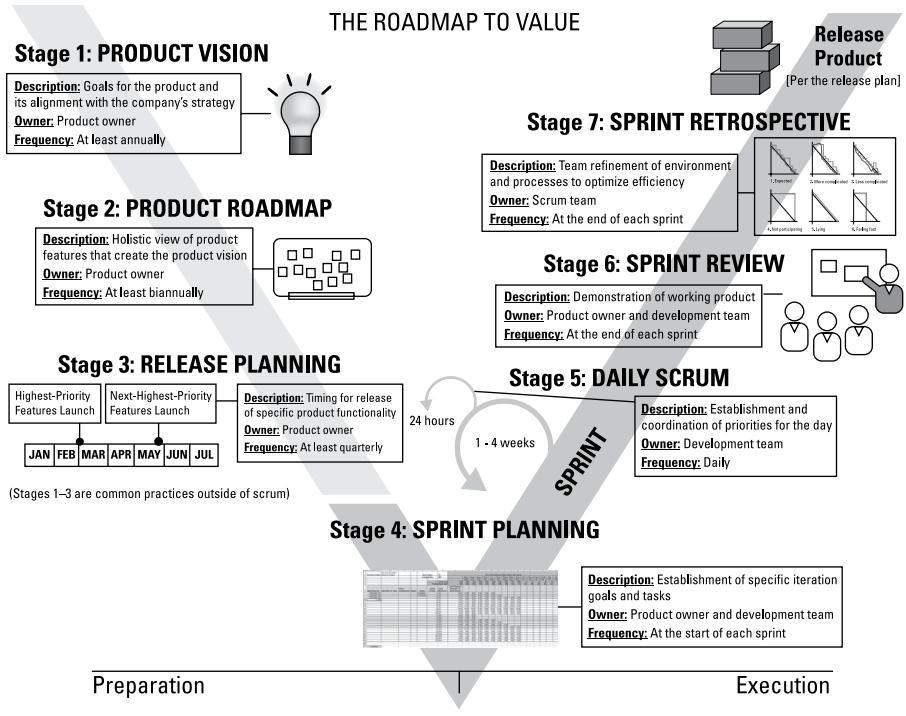


FIGURE 9-2:
Stages of agile planning and execution with the Roadmap to Value.

- » 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. Many releases may be required to accomplish the product vision and the highest-priority features should appear first. You create a release plan at the beginning of each release, which according to Principle 3 should be "frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale." Read more about release planning in Chapter 10.
- » 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. Read more about sprint planning in Chapter 10.

- » In stage 5, the development team has *daily scrum* meetings during each sprint to coordinate the day's priorities for accomplishing the sprint goal. In the daily scrum meeting, based on what was completed up to that point, you coordinate what you will work on today and any roadblocks, so that you can address issues immediately. Read about daily scrums in Chapter 11.
- » In stage 6, the scrum team holds a *sprint review* at the end of every sprint. In the sprint review, you demonstrate the working product to the product stakeholders. Find out how to conduct sprint reviews in Chapter 12.
- » 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 12.

Each stage of the Roadmap to Value is repeatable and contains planning activities. Agile planning, like agile development, is iterative and barely sufficient.

Progressive elaboration

During each stage of product development, you plan only as much as you need to plan. In the early stages of your work, 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 product development without disrupting the flow.

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



REMEMBER

Standish Group studies show that customers rarely or never use as much as 80 percent of the features in an application. In the first few development cycles of an agile product development effort, 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 development, 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.
- » 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.

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



At the beginning of development, 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. Remember the Agile Manifesto value: "Responding to change over following a plan."

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

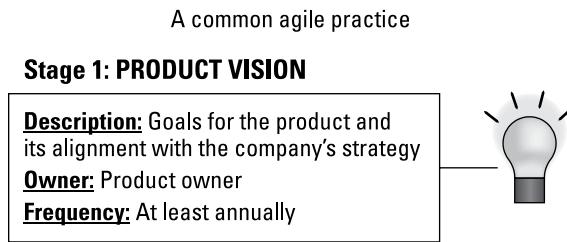
Defining the Product Vision

The first stage in the Roadmap to Value 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. As well-known author and spokesperson Simon Sinek explains, this is your "Why."

Figure 9-3 shows how the vision statement — stage 1 of the Roadmap to Value — fits with the rest of the stages and activities in product development.

FIGURE 9-3:
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 development. 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 product owner, development team, scrum master, and stakeholders refer to throughout their work.

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 stakeholders and revise it based on feedback.**
- 4. Finalize the product vision statement.**

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

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:

- » **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? The product canvas discussed in Chapter 4 is helpful for defining product goals.
- » **Customer:** Who will use the product? This question might have more than one answer.
- » **Need:** Why does the customer need the product? What features are critical to the customer? What problem will the product solve, as discussed in Chapter 4?
- » **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? An effectively written product vision statement can start you on the path to quickly learning the answers to these questions.



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 9-4, 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.

FIGURE 9-4:
Expansion of
Moore's template
for a vision
statement.

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)



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 more money” or “make customers happy” or “sell more products.” You want the vision statement to help you make scope decisions throughout product development. 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 vision statement compelling?
- » Is the vision concise?

These yes-or-no questions will help you determine whether your vision statement is thorough. 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:

- » **Product stakeholders:** The stakeholders will be able to identify that the vision statement includes everything the product should accomplish.
- » **Your development team:** The people who will be creating the product must also understand and have ownership of what the product needs to accomplish. Many product owners create the product vision with the development team, aligning purpose and motivation (Principle 5).
- » **Scrum master:** A strong understanding of the product will help the scrum master proactively remove roadblocks, enabling the team to accomplish the product vision.
- » **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 stakeholders, development team, and scrum master fully understand the statement.

Step 4: Finalizing the vision statement

After you finish revising the vision statement, make sure the 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 product.

If your product development will be more than a year long, you may want to revisit the vision statement. 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 product, product development investment should end when the vision is achieved and expansion of the vision is no longer viable.



REMEMBER

The product owner owns the product vision statement and is responsible for its preparation and communication across and outside 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 initial definition of strategy and desired value outcome in your agile product development. 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 9–5), 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 sequence for releasing to the customer.

A common agile practice

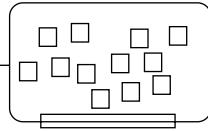
FIGURE 9-5:
The product
roadmap as part
of the Roadmap
to Value.

Stage 2: PRODUCT ROADMAP

Description: Holistic view of product features that create the product vision

Owner: Product owner

Frequency: At least semiannually



As with the product vision statement, the product owner creates the product roadmap with help from the development team and stakeholders. The development team often 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 development. In the product roadmap phase, it’s okay for your requirements detail, estimates, and time frames to be at a very high level.

To create your product roadmap, you do the following:

- 1. Identify stakeholders.**
- 2. List product requirements and visualize them.**
- 3. Arrange the product requirements based on value, risk, 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, plan to update your product roadmap at least twice a year.



TIP

Your product roadmap can be as simple as sticky notes arranged on a physical or virtual 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 product 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 product.

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, the product 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. Plus, your customer service representatives actually talk to your end users on a daily basis and probably have many insights you should consider.
- » **Sales department:** Make sure that the sales team sees the product so that it starts selling the same thing you're 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 product at risk if discovered later.
- » **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 product development, 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 10.

(continued)

(continued)

- **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 10.
- **Tasks:** *Tasks* are the execution steps required to develop a requirement into working functionality and generally mirror your definition of done as well as the tasks necessary to accomplish the story's acceptance criteria. You can find out about tasks and sprint planning in Chapter 10.

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 product discovery workshop, where the stakeholders and the development team meet and write down as many requirements as possible. Each item should be written in the customer's words rather than using technical jargon. For example, you might begin each item with the words, "My customer can now . . ." to reinforce how the requirements should relate to the customer, their problem, and the product vision. For example:

My customer can now

See her account balance

Pay bills

See her latest transactions

Provide feedback

Get help



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 allow 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 requirement, 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
- » Integration with other accounts

Figure 9–6 shows features grouped by themes.

Common activities	Reduction in call volume
Authenticate and access my accounts	Pay bills
View balance	Order checks
Transfer money between accounts	Order a copy of a statement
View pending transactions	Put a stop on a check or range of checks
View a statement	Open an account
View bills	Change password
Find a branch/ATM machine	Call customer service

FIGURE 9-6:
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 customer and therefore the organization creating that product.
- » *Risk* refers to the negative effect a requirement can have due to customer uncertainty or on product development.



TIP

You can estimate and prioritize requirements of any estimate size 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 we will implement them. Forcing an order requires making a priority decision for every requirement relative to every other requirement. A scrum team works on one thing at a time, so it's important to format your product roadmap accordingly. In Chapter 13, additional prioritization techniques are discussed.

To estimate and assign effort values to your requirements, you work with two different groups of people:

- » The development team determines the effort to implement the functionality for each requirement. Only the people who will do the work should provide effort estimates. The development team also provides critical feedback to the product owner for understanding how technical risks affect the ordering of the product backlog.
- » 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 10, we show you relative estimation techniques that scrum 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 traditional 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 product at the beginning?

Relative estimating is a self-correcting mechanism that allows scrum 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. Product development teams value accuracy over precision.

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 have an application that requires users to set up a user profile, they'll need to be authenticated by a username and password. The requirement for creating the username and password would be dependent on setting up a profile because you generally need a username and password to establish a user profile.

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 effect 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 risk. If a high-risk item will cause the product or its development to fail (an issue that cannot be resolved), scrum teams learn about it early. If something is going to fail, you want to fail early, fail cheap, and move on to a new opportunity that has value. In that sense, failure is a form of success for a scrum 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 towards the bottom of the roadmap or, better yet, be removed. If anything on your roadmap does not support the fulfillment of your product vision, you may want to ask whether it is truly needed. Remember Principle 10, "Simplicity — the art of maximizing the amount of work not done — is essential."



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 functionalities 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 9–7.

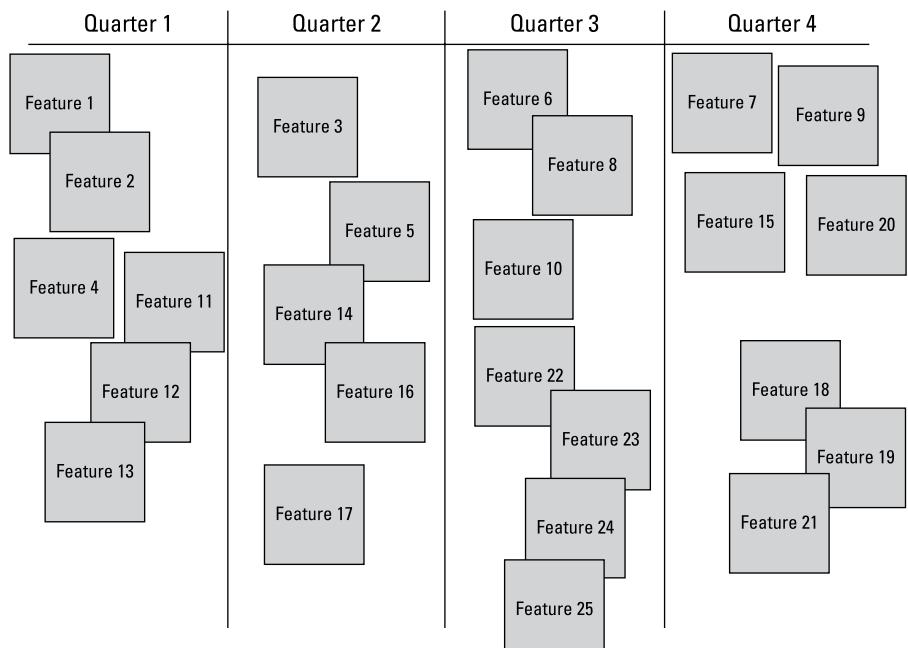


FIGURE 9-7:
Product roadmap
with ordered
requirements.

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 product development.

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 product development, 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 every team we've worked with has a product vision, product roadmap, and release plan for its first release, and is ready to start its sprint in as little as two to three days! To begin developing the product, you need only enough requirements for your first sprint. Having bought yourself some time, you can determine the rest as the team learns more from reality through progressive elaboration.

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. Whatever format you choose, ensure that the roadmap can be easily changed and transparently accessed.

You update the product roadmap throughout development, as priorities change. For now, the contents of the first release should be clear — and that's all you need to worry about to start executing and delivering value.

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 product. The product owner is responsible for creating and maintaining the product backlog by updating (adding, changing, removing) and prioritizing requirements. The scrum team uses the prioritized product backlog throughout development to plan its work each release and each sprint.

Figure 9–8 shows a sample product backlog. At a minimum, when creating your product backlog, be sure to do the following:

- » Include a description of your 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 9–8:
Product backlog items sample.

We also like to include the type of backlog item as well as the status. Teams will work mainly on developing features as described in the words of the user (user stories). But there may be a need for other types of product backlog items, such as overhead items (things the 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 9–8. The product owner prioritizes all product backlog items through the lens of the customer and stakeholders.



REMEMBER

In Chapter 2, we explain how documents for agile product development 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 product development.

The scrum team refers to the product backlog as the main source for requirements. If a requirement exists, it's in the product backlog.

The user stories in your product backlog will change throughout development in several ways. For example, as the team completes user stories, you mark those stories as complete in the backlog. You also record any new user stories. Some user stories 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 user stories 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. See Chapter 15 for more about using the product backlog estimate to predict the release length and cost.



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 describe in the next chapter.

IN THIS CHAPTER

- » Decomposing requirements and creating user stories
- » Creating a product backlog, release plan, and sprint backlog
- » Getting the product ready to ship and preparing the rest of the organization for the release
- » Making sure the marketplace is ready

Chapter **10**

Planning Releases and Sprints

After you create a product roadmap for your product (see Chapter 9) 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. We also discuss how to prepare the rest of the organization for the release, including operational support, and ensuring that the marketplace will be ready for your release.

First, you see how to break down the larger requirements from your product roadmap into smaller, more manageable requirements called *user stories*.

Refining Requirements and Estimates

You start agile development with very large requirements. As the work progresses and you get closer to developing those requirements, you will break them down into smaller parts — small enough to begin developing. See Chapter 9 for more about this process, which is known as *decomposition*.

One clear, effective pattern for defining product requirements is the user story. 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. It's called a *story* because the simplest way people tell stories is by talking to each other. User stories are most effective when they're told by speaking to each other face-to-face. The written pattern described in this section can be used to help with that conversation.

Traditional software 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 through the user story pattern, its work becomes more personal and real. The team comes to know the benefit of each requirement to the user (or the customer or the business) 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 tells the "who," "what," and "why" for the desired functionality. The user story should also be supported by a list of validation steps (*acceptance criteria*) to take so you can confirm whether the working requirement for the user story is correct. Acceptance criteria follows this pattern:

When I (take this action), (this happens)

User stories may also include the following:

- » **A user story ID:** A unique identifying number to differentiate this user story from other user stories in the product backlog tracking system.
- » **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 9.

» **The name of the person who thought of the user story:** Anyone can create a user story.



TIP

Although agile product development approaches encourage low-tech tools, the scrum team should also find out what works best 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 the simplicity of index cards, but use what works best for your scrum team and your product.

Figure 10-1 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.

FIGURE 10-1:
Card-based user story example.

Title	Transfer money between accounts		
As	Carol,		
I want to	transfer funds between accounts		
so that	each account has the correct amount of funds		
Value	Jennifer	Author	Estimate

When I do this:	This happens:
When I view my account balances,	I see an option to transfer funds.
When I select the transfer option,	I choose between which accounts I want to transfer funds.
When I select the “transfer from” option,	I see a list of my available accounts and balances.
When I select the “transfer to” option,	I see a list of my available accounts and balances.



TIP

The three Cs formula for user story creation — card, conversation, and confirmation — illustrates how the user story pattern enables scrum teams to create customer value. By limiting user stories to fit on a 3x5 index card, we encourage a *conversation* for achieving a shared understanding of the job to be done for the customer (rather than excessive documentation that implies nothing is left to discuss). If the conversation is supported by the answers to the quiz up front (acceptance criteria that is a *confirmation* that the actions the user will take meet the intended needs), you’re probably on the right track.

The product owner gathers the user stories and manages them (that is, determines the priority and initiates the decomposition discussions). It is not the sole responsibility of the product owner to write user stories. The development team and other stakeholders are also involved in creating and decomposing user stories to ensure clarity and shared understanding across the scrum team.



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're very effective in conveying exactly what a requirement needs to do for the customer.

The big benefit of the user story pattern is realized 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 its intention. Using the user's voice is something all can understand and relate to — not so much with technical jargon.

We use user stories as examples of requirements for software product development 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 and other product types.

Steps to create a user story

When creating a user story, follow these steps:

1. Identify the stakeholders.
2. Identify who will use the product.
3. Working with the stakeholders, write down, in a user story format, what the product will need to do.

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 items to your product backlog later. The best changes often come at the end, when you know the most about the product and the end-customers.

Identifying product stakeholders

You probably have a good idea about who your stakeholders are — anyone involved with, affected by, or who can affect the product and its creation. Stakeholders provide valuable feedback about every product increment you deliver each sprint.



REMEMBER

You will work with stakeholders also when you create the product vision and product roadmap.

Make sure the stakeholders are available to help you gather and write product backlog items. Stakeholders of the sample mobile banking application introduced in Chapter 9 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 stakeholders.
- » Users of your product.
- » 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 product.
- » Technical stakeholders. These are people who work with the systems that might need to interact with your product.

Identifying users

As discussed in Chapter 4, agile product development is customer focused. Building on the personas you've defined, your understanding of their needs, and the problems to be solved helps the team more clearly understand the product requirements. Knowing who your end users are and how they will interact with your product drives how you define and implement each item 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 categories 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 hypothetical person. For instance, "Ellen is a 65-year-old retired engineer who is spending her retirement traveling the world. Her net worth is \$1,000,000, and she has residual income from several investment real estate properties." Read more about personas in Chapter 4.

Ellen represents 30 percent of XYZ Bank's customers, and a good portion of the product roadmap includes features that someone like Ellen will use. Instead of repeating all the details about Ellen every time the scrum team discusses these features, the team can simply refer to the type of user as Ellen. The product owner might identify several personas, as needed, and will even print the descriptions with a stock photo of what Ellen might look like and post them on the wall in the team's work area to refer to throughout development.



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 product. 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 users, 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 product discovery workshop. See more about product discovery workshops in Chapter 4.

Have the stakeholders write down as many requirements as they can think of, using the user story format. One user story for the product and personas from the previous 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 decide whether I have enough money in my account to make a transaction

» Back side of card:

- **When** I sign into the XYZ Bank mobile application, my checking account balance appears.
- **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 10–2.



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.

You will create new user stories throughout product development. You'll also take existing large requirements and decompose them until they're manageable enough to work on during a sprint.

Title	Transfer money between accounts
As	Carol,
I want to	categorize expenses,
so that	I can easily identify my purchases made for my clients.
<hr/> Value	<hr/> Jennifer Author
	<hr/> Estimate

Title	Put stop on a check
As	Nick,
I want to	stop payment on a lost or stolen check,
so that	I can avoid any unauthorized activity on my account.
<hr/> Value	<hr/> Caroline Author
	<hr/> Estimate

FIGURE 10-2:
Sample user stories.

Breaking down requirements

You refine requirements many times throughout development. For example:

- » When you create the product roadmap (see Chapter 9), 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 (see the “Estimation poker” section later in the chapter to learn about Fibonacci sizing). Both features and themes are considered large by a development team.
- » 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 break down requirements even further. User stories are broken down to eight points or fewer. See Figure 10-3 for a helpful requirement decomposition guide.

User Story	Epic	Feature
Sprint	Release	Roadmap
1 2 3 5 8	13 21 34	55 89 144
XS S M L XL		

FIGURE 10-3:
User story decomposition guidelines.

To decompose requirements, you'll want to think about how to break down the requirement into individual actions. Table 10-1 shows a requirement from the XYZ Bank application introduced in Chapter 9 that is decomposed from the theme level down to the user story level.

TABLE 10-1 Decomposing a Requirement

Requirement Level	Requirement
Theme	See bank account data on a mobile device.
Features	See account balances. See a list of recent withdrawals or purchases. See a list of recent deposits. See my upcoming automatic bill payments. See my account alerts.
Epic user stories — decomposed from “see account balances”	See checking account balance. See savings account balance. See loan balance. See investment account balance. See retirement account balance.
User stories — decomposed from “see checking account balance”	See a list of my accounts once securely logged in. Select and view my checking account. See account balance changes after withdrawals. See account balance changes after purchases. See day's end account balance. See available account balance. Change account view.

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 user story should need no other user 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 user story demonstrates product value to the customer. It describes features, not technical tasks to implement it. The user story is in the user's language and is easy to explain. The people using the product or system can understand the user 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 of the work required to complete your user stories 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 these stories describe.

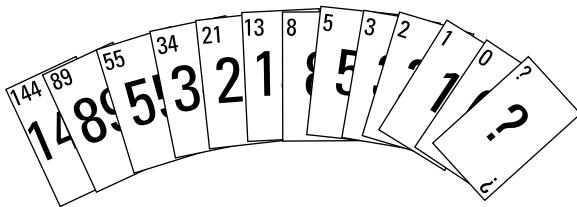
To play estimation poker, you need a deck of cards like the one in Figure 10-4. You can get a digital version online at our website (www.platinumedge.com/estimationpoker), or you can make your own with index cards and markers. The

numbers on the cards are from the Fibonacci sequence, which follows this progression:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, and so on

If we start with the numbers 1 and 2, each subsequent number in the Fibonacci sequence is derived by taking the sum of the previous two numbers.

FIGURE 10-4:
A deck of estimation poker cards.



Each user story receives an estimate relative to other user stories. For instance, a user story that is a 5 requires more effort 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.**

This user story is the *anchor user story* for the team. The scrum master helps the development team reach consensus by using a fist of five or thumbs up/thumbs down (as described in Chapter 7), with discussion until everyone agrees on a user story that represents an estimate of 5.

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.

You don't want the players to see each other's cards until all cards have been played. This limits how much one player can influence others to vote a certain way. 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.)

5. All players turn over their cards simultaneously.

6. If the players have different story points:

a. It's time for discussion.

Discussion is the value-add of estimation poker, which enables team alignment and consensus. 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 a fist of five or thumbs up/thumbs down, as described in Chapter 7) 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 (including test automation), 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 may spend about 10 percent of their time each sprint decomposing and refining product backlog items, including estimating and re-estimating. 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 estimate your entire product roadmap, but one that allows you 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 divide 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 user 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.



REMEMBER

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.

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 spend 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 10-5.

SIZE	POINTS
Extra small (XS)	1
Small (S)	2
Medium (M)	3
Large (L)	5
Extra large (XL)	8

FIGURE 10-5:
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.

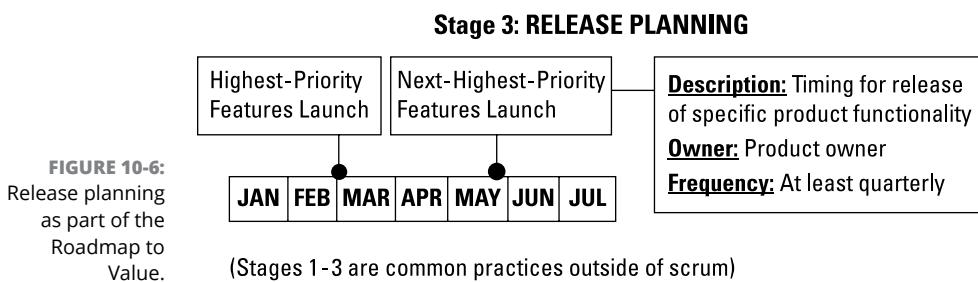
That's it. In a few hours, your entire product backlog was estimated. In addition, your scrum team has a shared understanding of what the requirements mean, having discussed them face-to-face rather than relying on interpretations of extensive documentation.

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 include highest priority (high value, or high risk, or both) items and exclude lower-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 9 to see the roadmap as a whole). Figure 10-6 shows how release planning fits into product development.



Release planning involves completing two key activities:

- » **Revising the product backlog:** In Chapter 9, we tell you that the product backlog is a comprehensive list of all the user stories you currently know for your product, whether or not they belong in the current release. Keep in mind that your list of user stories will probably change throughout development.
- » **Creating the release plan:** This activity consists of defining the release goal, release target date, and prioritization of product backlog items that support the release goal. Whereas the product vision provides the long-range goal of the product, 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 information radiators between the product owner and the development team. (See Chapter 11 for more on information radiators.) In Chapter 9, 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. The first case has a fixed scope and flexible date; the second case has a fixed date and a flexible scope.

If a fixed date and fixed scope are determined for the release, adjustments may need to be made to the number of teams to accomplish the release goal according to schedule. The development team, not the product owner, estimates the effort required to implement product backlog items. Imposing a fixed scope and timeline without adjusting quality or resources (in this case, talent resources) will not be successful.

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 a release goal 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. This approach provides appropriate flexibility and slack for the scrum team to deliver value without having to complete every single task.

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 goal is sized. (Refer to Figure 10-3.) We like to make sure that items supporting the current release goal have been decomposed and are sized appropriately for the release. The development team helps the product owner by updating estimates for any added or revised user stories, and commits to the release goal with the product owner.



TIP

Release planning is the initial opportunity to identify and break down dependencies before they become impediments. Dependencies are anti-patterns to becoming agile. Teams should work to become highly aligned and highly autonomous. Dependencies are an indication that your team does not have the capability to do whatever it is dependent on.

5. Estimate the number of sprints needed, based on the scrum team's velocity.



TECHNICAL STUFF

Scrum teams use velocity as an input 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, its 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 15.

Be aware that some teams add a *release sprint* to a release to conduct activities 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.



REMEMBER

Delaying crucial development tasks such as testing until the end of development rearloads risk. Agile techniques frontload risk to avoid the surprises and defects that result from delayed testing. If a scrum team requires a release sprint, it probably means the broader organization can't support being truly shippable each sprint, which is an impediment to becoming agile. The goal for scrum teams is to have every type of work or activity required to release functionality to the market as part of the sprint-level definition of done. Scrum masters should work together to remove organizational impediments preventing teams from being able to release at scale according to their sprint-level definition of done.

In some traditional or project-focused organizations, some tasks, such as security testing or load testing a software product, can't be completed within a sprint because the task's environment takes 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. In these examples, the scrum master would work with the organizational leaders who manage the security or load testing environments to find ways to enable scrum teams to accomplish security or lead testing during the sprint.

Each planned release shifts from what was a tentative plan (high-level product roadmap items) to a more concrete goal ready to execute in the sprint(s) of the release. Figure 10-7 represents a typical release plan.

Release Goal: Enable customers to access, view, and transact against their active accounts
Release Date: March 31, 2021

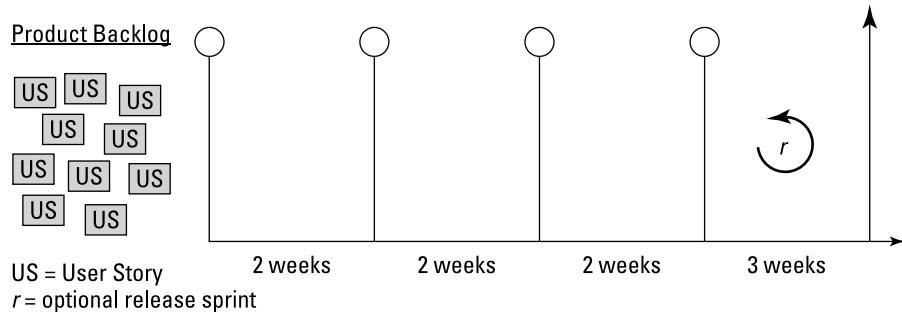


FIGURE 10-7:
Sample release plan.



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?

Preparing for Release

In release planning, you also need to prepare your organization for the product release. The next section discusses 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 the product for deployment

In each sprint, a valuable working product increment is created in support of the release goal. With a definition of done that means each sprint's increment is shippable, there's not much more you need to do to prepare the product for technical deployment. Every sprint, you're ready to release if there is enough value accumulated for the customer.



TECHNICAL STUFF

With software product development, releasing to production is done through continuous integration (CI) and continuous deployment (CD), which are extreme programming (XP) practices used with software product development. (You can read more about CI in Chapter 17.) The product code is checked in and moved to quality assurance (QA) and then to the production (live) environments as quickly and seamlessly as possible. Advancements in technology enable teams to automate a pipeline for building, integrating, testing, and fixing without delay. Combining a CI/CD pipeline with a robust automated testing harness raises the bar of your product development agility. Teams developing non-software products should use techniques that automate testing and integration of new functionality to the existing product as much as possible.



TECHNICAL STUFF

In information technology (IT), usually involving software development, *development operations (DevOps)* is the integration of software development and IT operations (which includes functions such as systems administration and server maintenance). Taking a DevOps approach enables everyone involved (user experience, testing, infrastructure, database, coding, design) to work together, to eliminate handoffs, and to streamline collaboration for reduced deployment cycle times. Multiple teams working on the same product will not be successful without a reliable CI/CD pipeline.



TIP

Not all agile product development efforts 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 product's technological complexity can all help determine your approach to product releases. Helpful for this discussion are Principles 1 and 3, respectively: "Satisfy the customer through early and continuous delivery" and "Deliver...frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale." (For details, see Chapter 2.)

Prepare for operational support

After your product is released to the customer, someone will have to support it, a responsibility that 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.

We recommend a model that separates new development and maintenance work, as illustrated in Figure 10-8.

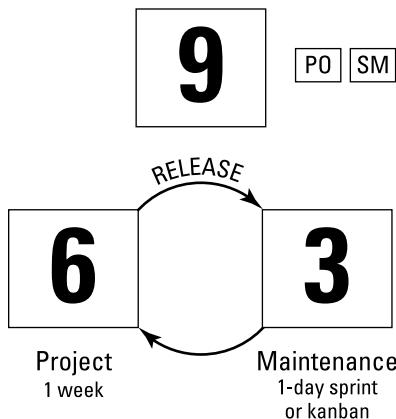


FIGURE 10-8:
Operational support scrum team model.

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 work from the product backlog in one-week to two-week sprints, as described in Chapters 9 through 12. The work that the team commits to during the sprint planning meeting will be the only work it does.

The members of the three-person team are our firefighters and do maintenance and support work in one-day sprints or by using kanban. (You learn about one-day sprints in Chapter 11 and kanban in Chapter 5.) 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 product 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.

Teams should rotate team members between the two activities at sprint boundaries (every three to five sprints, for example) to give everyone an opportunity to learn from both types of work. If support is excessive, the product owner may want to reevaluate the product backlog to see if there are ways to reduce the weight of support and its distraction. Support distractions cause the team to focus on tactical resolutions rather than strategic value creation. Team stability and acceleration will improve.

When preparing for release, establishing expectations up front regarding 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's deployed. Establishing expectations also increases ownership across the scrum team and heightens the team's awareness and dedication to long-term success.

Product owners who maintain a strong working relationship with help desks or those providing customer support benefit greatly by understanding how their products are used by real users. Help desk reporting can be valuable for evaluating product backlog candidates or upcoming priorities. Help desks benefit by knowing that the scrum team is working to address any escalated incidents. The product owner involves these groups in release planning to ensure that all are prepared for operational support well in advance of release.

Preparing the organization

A product release often affects a number of departments in a company or an organization. To get the organization ready for the new functionality to be released next, the product owner coordinates with the rest of the organization regarding what to expect and what will be needed from them during release planning. When product owners do this effectively, there shouldn't be surprises at release time.

Release planning addresses not only the activities for the development team to release but also the activities to be performed by the rest of the organization. These might include the following:

- » **Marketing:** Do marketing campaigns related to the new product need to launch at the same time as the product?
- » **Sales:** Do specific customers 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 release and the specific tasks these groups need to complete will 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.

During release planning, you also need to include one more group: the customer. The next section discusses getting the marketplace ready for your product.

Preparing the marketplace

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 as to the release date and the features that will be part of the release.



REMEMBER

Some software products are for only internal employee use. Certain things you're reading in this section might seem like overkill for an internal application — that is, 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 is 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 to get real-world feedback about the product from a subset of end users. (Some people use focus groups.) Your marketing team can also use this feedback to 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 items on your release 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 4.

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!

Sprint Planning

With agile product development, 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. 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. One month 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, which structurally prevents weekend work. When priorities change on a daily basis, some scrum teams work in one-day sprints, as discussed in Chapter 11.

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. Sprint duration is a function of the business's need for change.

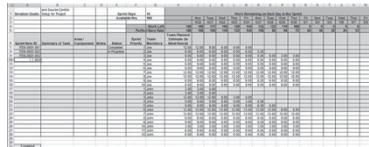
Each sprint includes the following:

- » Sprint planning at the beginning of the sprint
- » Daily scrum meetings
- » Development work — the bulk of the sprint
- » A sprint review and a sprint retrospective at the end of the sprint

Discover more about daily scrums, development work, sprint reviews, and sprint retrospectives in Chapters 11 and 12.

Sprint planning is stage 4 in the Roadmap to Value, as you can see in Figure 10–9. The entire scrum team — the product owner, scrum master, and 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 10-9:
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 the goal for your sprint.
- » Choose the product backlog items (user stories) that support your goal.
- » Break user stories into specific development tasks.
- » Create a sprint backlog. The *sprint backlog* consists of the following:
 - The list of user stories in the sprint in order of priority.
 - The relative effort estimate for each user story.
 - The tasks necessary to develop each user story.



TIP

- The effort, in hours, to complete each task (if needed). At the task level, if you estimate the number of hours each task will take to complete, use hours instead of using story points. Because your sprint has a specific short length, and thus a known number of available working hours, you can use the time each task takes to determine whether the tasks will fit into the team's capacity of the sprint. Each task should take one day or less for the development team to complete.

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 in progress for the sprint.



TECHNICAL STUFF

Tasks 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're 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 development targets 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 and the scrum master should investigate what might be keeping the team member from finishing work. (For more on managing roadblocks, see Chapter 11.)

The development team collaborates to create and maintain the sprint backlog, but 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 10-10 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.

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.

MyXYZ Mobile Banking App – Sprint 1									
Sprint Dates: February 4 – February 8									
Sprint Goal	Available working hours in the sprint						Days in sprint :	4.5	Total
Demonstrate the ability for personal banking users to view account balances and pending translations.	Developers (Start w/ sprint planning)						(End w/ sprint review + retrospective)		
	M	Tu	W	Th	F				
	4	5	6	7	8				
	Suraj	4	6	6	6	4	26		
	Nancy	4	6	6	6	4	26		
	Kavita	4	6	6	6	4	26		
	Liam	4	6	6	6	4	26		
	Paul	4	6	6	6	4	26		
								Total sprint hours:	130
								Total per day:	29
Burndown: Hours & Points Remaining									
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	View account balance	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	View pending transactions	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 10-10:
Sprint backlog example.

Base the length of your sprint planning meeting on the length of your sprints. Sprint planning should take no longer than two hours per week of the sprint, or no longer than a full day for a one-month sprint. This timebox helps ensure that the meeting stays focused and on track. Figure 10-11 is a good quick reference for your sprint planning meeting lengths.

If my sprint is this long...	My sprint planning meeting should last no longer than...
One week	Two hours
Two weeks	Four hours
Three weeks	Six hours
Four weeks	Eight hours

FIGURE 10-11:
Ratio of sprint planning meeting to sprint length.



TECHNICAL STUFF

With agile product development, the practice of limiting the time of your meetings is sometimes called *timeboxing*. Keeping your meetings timeboxed provides focus and 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, decide what to do during the sprint by doing the following:

- 1. Discuss and set a sprint goal.**
- 2. Select the user stories from the product backlog that support the sprint goal, further refine them for understanding, 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.**



TIP

Consistently refining the product backlog ensures that the scrum team plans items into each sprint with which the team is already familiar. This refinement is also critical for ensuring that sprint planning stays within the timebox and results in a clear plan to deliver potentially shippable functionality at the end of each sprint. Scrum teams, on average, spend about 10 percent of their sprint in product backlog refinement for future sprints.

At the beginning of your sprint planning meeting, the product owner should propose a sprint goal that identifies a problem to solve for the customer 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 9) 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 and refine the estimates for those user stories, as needed. For the mobile banking

application sprint goal example, 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.



REMEMBER

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, 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 on 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. What you learn in sprint 1 may fundamentally change how you go about sprint 2 or 10 or 100.

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 (including test automation), 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.
- » Write the user acceptance 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/Android application. (This task could potentially be three different tasks.)
- » Create calls to the database to verify the username and password.

- » Re-factor code for mobile devices.
- » Write the integration test.
- » Promote the product increment to QA.
- » Update the regression test automation suite.
- » Run the security 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 during the 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 product development.
- » **Sprint 2:** Assuming the scrum team was able to complete sprint 1 successfully, 50 percent of what the development team thinks it can accomplish.
- » **Sprint 3:** Assuming success in sprint 2, 75 percent of what the development team thinks it can accomplish.
- » **Sprint 4 and forward:** Assuming success in sprint 3, 90 percent. The development team will have developed a rhythm and velocity, gained insight into agile principles and the product, and will be working at close to full pace.



TIP

Avoid planning 100 percent of capacity for a sprint. Scrum teams should build in slack in their sprint to account for unknowns that inevitably come up. Instead of padding estimates, simply be wise and don't commit every available hour, assuming everything will go as planned. Teams that finish early accelerate faster.

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 12). This evaluation is especially important for the first sprint.



TIP

For the sprint, how many total working hours are available? In a one-week sprint or a 40-hour week, you could wisely assume that 4.5 working days are available to develop user stories. Why 4.5 days? About one-fourth of day one is taken up with sprint planning, and about one-fourth of day five is taken up with the sprint review (when the stakeholders review the completed work) and the sprint retrospective (when the scrum team identifies team improvements for future sprints). That leaves 4.5 days of development. If you assume each full-time team member has 30 hours per week (6 productive hours per day) to focus on the sprint goal, the number of working hours available is

$$\text{Number of team members} \times 6 \text{ hours} \times 4.5 \text{ days}$$

After 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 vision and roadmap, product backlog, definition of done, and sprint backlog are in a prominent place and accessible to everyone during sprint planning as well as in the area in which they work. In this way, stakeholders can view the product information and progress on demand without interrupting the development team. For details, see Chapter 11.

IN THIS CHAPTER

- » Planning each day
- » Tracking daily progress
- » Developing and testing every day
- » Ending the day

Chapter **11**

Working throughout 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, inspecting and adapting, 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 ensure transparency as they help create the product.

Planning Your Day: The Daily Scrum

With agile product development, you make plans throughout the entire development effort — and on a daily basis. Agile development teams start each workday with a *daily scrum* meeting to evaluate their progress and adapt their plan for the day based on what was accomplished previously. They identify and coordinate the resolution of impediments (roadblocks requiring scrum master involvement),

note completed items, and 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 product development in Figure 11-1. Note how they both repeat.

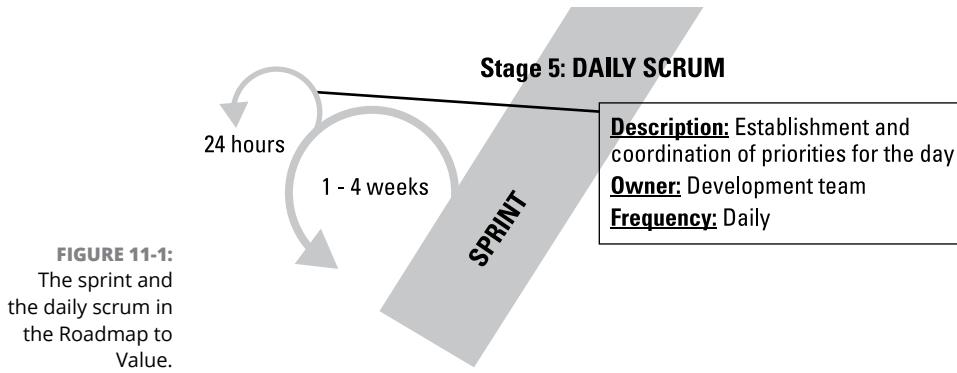


FIGURE 11-1:
The sprint and
the daily scrum in
the Roadmap to
Value.

In the daily scrum meeting, each development team member addresses the following four topics, which facilitate team coordination:



WARNING

» **What was completed yesterday** to help accomplish the sprint goal?

Avoid using the daily scrum as a status report by having each developer give an accounting of what they did the day before or by moving completed items around on the task board. Developers should update their task as soon as they complete it, or at the very least at the end of the day, so that when the scrum team comes together for its daily scrum the next day, the status is already reflected. In other words, don't spend time on what was accomplished yesterday unless it effects how to go about the work to be done today.

» **What will be done today** to help accomplish the sprint goal?

» **What impediments** are in the way of accomplishing the sprint goal?

» **This is how I feel.** (We added the fourth question to help the scrum master better understand team health daily rather than once per sprint.)



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. Daily scrum is how scrum refers to it.

We also have the scrum master address these three statements regarding the team's impediments:

- » Impediments resolved yesterday
- » Impediments that need to be resolved today (and order of priority)
- » Impediments that need to be escalated

What does the product owner do during the daily scrum? Listen. The product owner listens to see if there is anything he or she can do to help the team accomplish its work more effectively. The product owner may provide clarification as needed, and might say something if he or she hears something that indicates that the development team is working on something outside the sprint goal. An engaged, decisive product owner makes life easier for a development team.

One of the rules of scrum is that the daily scrum meeting should last 15 minutes or less. Longer meetings eat into the development team's day. Standing encourages shorter meetings (which is why the meeting is referred to also as the daily standup). You can also use props to keep daily scrum meetings quick.



TIP

We start meetings by tossing an item, such as a squeaky burger-shaped dog toy — don't worry; it's clean — to a random development team member. Each person addresses the four topics 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 distract the development team from the focus of the sprint.
- » **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 having relevant discussions of the tasks they're working on and 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, and should take developers about one minute.

» **Such a short meeting must start on time.** It's not unusual for the scrum team to have a working agreement ensuring that meetings start and end on time. Creative punishments for tardiness include doing pushups or adding penalty money into a team celebration fund. Whatever punishment 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.

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 30 minutes 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 daily scrum meeting also allows the development team time to review defect reports from automated testing tools that were run the night or weekend 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 needed. 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 11-2 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 10 discusses details of the sprint backlog.)

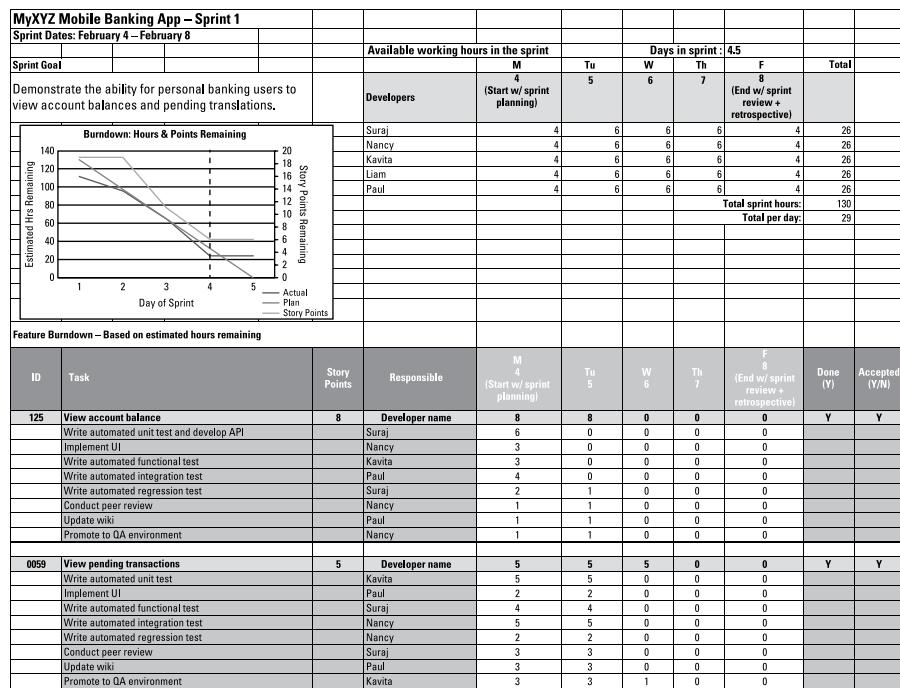


FIGURE 11-2:
Sample sprint backlog.

Make the sprint backlog available to the entire team every day. That way, anyone who needs to know the sprint status can find it instantly.

Near the top left of Figure 11-2, 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 11-3 shows the burndown chart in detail.

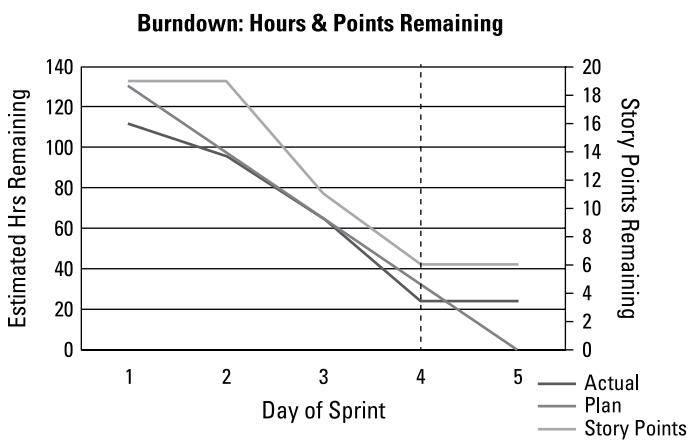


FIGURE 11-3:
A burndown
chart.

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 11-3, 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 to see the status of the sprint at a glance. 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 sprint goal and helps you make informed decisions early in the sprint.



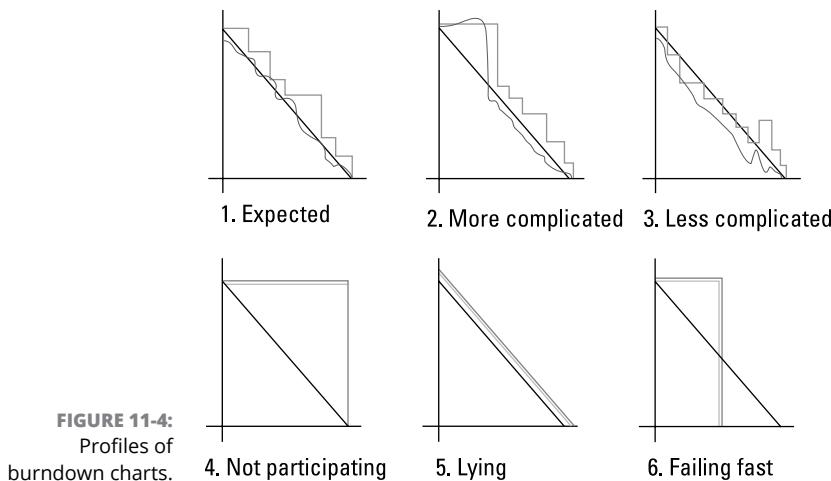
You can create a sprint backlog using a spreadsheet and charting program such as Microsoft Excel. You can also download our free sprint backlog template, which includes a burndown chart, from the book's website at www.dummies.com/go/agileprojectmanagementfd3e.

Figure 11-4 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.
- 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.



WARNING



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 12). 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 development 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 (not swarming). 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 the story 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. High-performing teams and organizations do one item well before moving onto the next item.

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 11-5 shows a typical task board. As you can see, the task board is a strong visual representation of the work in progress.



TECHNICAL
STUFF

The task board is a lot like a kanban board. *Kanban* is a Japanese term that means *visual signal*. Toyota created these boards as part of its lean manufacturing process.

In Figure 11-5, 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.

RELEASE GOAL:	SPRINT GOAL:		
RELEASE DATE:	SPRINT REVIEW:		
TO DO	IN PROGRESS	ACCEPT	DONE
			<div style="text-align: center;"> US Task Task Task </div>
		US 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 11-5:
Sample task board.

Agile product development day-to-day work 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.



REMEMBER

The product owner owns the product backlog. The development team owns the sprint backlog. Ownership means keeping the backlog updated, clarified, and transparent.

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 responsibilities of each member of the product team during the sprint.

Keys for daily product owner success

A successful product owner focuses on ensuring that the development team has everything it needs to keep the development speed high. The product owner works to understand the problems and needs of customers by meeting with them frequently. Product owners shield the development team from competing priorities, allowing the team to focus on the sprint goal. By sitting with the rest of the scrum team, the product owner can provide instant feedback as work is completed, enabling the development team to turn requirements into valuable working functionality.

The product owner has the following responsibilities during a typical day in a sprint:

» Proactive contributions:

- Look forward to the next sprint and elaborate user stories in readiness for the next backlog refinement or sprint planning meeting.
- Add new user stories to the product backlog as necessary and ensure that new user stories support the product vision, release goal, and sprint goal.
- Collaborate with other product owners or stakeholders to align on release or sprint goals. Maintain the product backlog as necessary.
- Review the product budget to stay abreast of product expenses and revenues.
- Review product performance information and trends in the marketplace.
- With the scrum master, watch for opportunities to proactively remove impediments that could impede development if not addressed early, such as product questions arising during sprint planning or legal wording.

» Reactive contributions:

- Provide immediate clarification and decisions about requirements to keep the development team developing.
- Remove business impediments brought by other members of the scrum team, such as unplanned requests from other teams or stakeholders. Shield the team from business distractions.
- Review completed user story functionality and provide feedback to the development team.

Keys for daily development team member success

Successful development team members have pride in their work. They build high-quality, enduring products. They engineer their work for change, realizing refactoring is necessary and expected as more is learned. They sit next to their teammates and perform tasks, even unfamiliar tasks, to expand their capabilities and contributions to their team. They excel in their particular discipline and look to expand their capability each day.

If you're a member of the development team, you

» Proactive contributions:

- Select the tasks of highest need and complete them as quickly as possible.
- 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.
- Collaborate with other scrum development teams to technically align on release or sprint goals.
- Continually improve regression test automation, CI/CD pipelines, and unit testing.
- Evaluate opportunities to improve the product architecture and development processes.
- Alert the scrum master to any roadblocks you can't effectively remove on your own.

» Reactive contributions:

- Request clarification from the product owner when you're unclear about a user story.
- 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 the amount of work remaining for your tasks in the sprint backlog.

Keys for daily scrum master success

Successful scrum masters are both coaches and facilitators. They coach the team to improved performance and facilitate team interactions to help the team reach decisions quickly. They inspire, lead, challenge, and serve. Because scrum masters also sit with the team, each day they look for opportunities to serve their team members by removing impediments, by coaching the broader organization to better work with the team, and by making sure the organization's environment enables success. After low-hanging-fruit team improvements are made, a scrum master's job only gets harder as he or she works to remove more difficult organizational impediments affecting the team.

If you're a scrum master, you do the following during a typical day:

» Proactive contributions:

- Uphold agile values and practices by coaching the product owner, development team, and organization when necessary.
- Remove roadblocks and organizational issues, both tactically for immediate problems and strategically for potential long-term issues. Scrum masters question the status quo of organizational constraints that strategically impede scrum teams from becoming higher functioning. In Chapter 7, we compare the scrum master to an aeronautical engineer, continually removing and preventing organizational drag on the development team.
- Build relationships to foster close cooperation with people working with the scrum team. Build clout and champion agility throughout the organization.

Nonverbal communication says a lot. Scrum masters can benefit from understanding body language to identify unspoken tensions in the scrum team.

- Prepare for upcoming facilitation opportunities, such as researching retrospective models to help the team maximize its retrospective discussion and acquiring supplies to facilitate affinity estimation.
- Shield the development team from external distractions.
- Collaborate with other scrum masters and stakeholders to resolve or escalate impediments.

» Reactive contribution:

- Facilitate consensus building in the scrum team, as needed.



TIP

We often tell scrum masters, "Never lunch alone. Always be building relationships." You never know when you'll need to rely on relationships to remove an obstacle.

Keys for daily stakeholder success

As members of the product team, successful stakeholders know how to work with the product owner to ensure product success. They counsel, collaborate, and listen. They give feedback and support. In flat, agile organizations, stakeholders empower, coach, and serve the scrum team rather than direct its activities from the outside or top-down. Each day, they participate in team discussions when asked but otherwise reserve their feedback for sprint review discussions. For more on the product stakeholder role, see Chapter 7.

If you're a product stakeholder, you do the following during a typical day:

» Proactive contributions:

- Counsel with the product owner on customer needs and backlog priorities.
- Look for opportunities to remove team impediments. Continually ask how you can help.

» Reactive contributions:

- Participate in sprint reviews and provide feedback. Be available to attend any other discussion requested by the team.
- Look at team burndowns or task boards. Look for opportunities to help the team become successful.
- Practice Principle 5, "Give the team the environment and support they need, then trust them to get the job done."

Keys for daily agile mentor success

For teams new to agile techniques, agile mentors are critical sounding boards for the team. They challenge the team's thinking, helping to create healthy tension. Similar to scrum masters, they coach, challenge, and serve. They teach them to find answers themselves (rather than just give them the answers). The team understands that it will get honesty and candor from the agile mentor. Agile mentors work to become redundant, transitioning their experience and expertise to the scrum master. Agile mentors participate daily in whatever way the team needs them.

Strategically, agile mentors engage with the organization's leaders to help maximize the value created by the teams. The scrum team's maximum pace is determined by the environment in which they're working. Agile mentors help leaders improve the environment according to agile values and principles. See Chapter 18 for more on the agile mentor's strategic role.

If you're an agile mentor, you do the following during a typical day:

» Proactive contributions:

- Counsel with the scrum master, primarily to help them build expertise, clout, and capability to effectively coach, trailblaze, and facilitate.
- Provide agility mentoring to developers and the product owner in the form of in-the-moment course corrections as they strive to learn and improve their roles.
- Coach stakeholders and other organizational leaders on how they can best support the scrum team to deliver valuable and potentially shippable functionality for the customer at every sprint.

» Reactive contributions:

- Observe scrum team events as well as informal interactions and provide feedback and guidance.
- Attend any discussion requested by the team.
- Inspect team burndowns or task boards. Provide feedback on opportunities to help the team become successful.

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 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 definition of done and is deemed ready to release. The development team may or may not release the increment at the end of the sprint because release timing depends on the release plan. The release plan 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

With agile product development, *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 products. 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 also sit with the development team. He or she focuses 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 5, 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 17.
- » **Avoid coding new features that are outside the sprint goal.** If new, nice-to-have features become apparent during development, add them to the product backlog.
- » **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. Continuous integration (CI) paired with continuous deployment (CD) is known as CI/CD. Together, teams are able to release early and often. Read more about CI/CD in Chapter 10.

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 testing for you. With automated testing, the development team can quickly develop and test the product, which is a big benefit for improved team agility.

Often, scrum teams develop during the day, with regression test automation and security vulnerability scanning run on a nightly or weekly cycle. After the cycle completes, the 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.

Software 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
- » **Integration testing:** Verifying that new functionality created in the development environment still works when integrated with the existing functionality
- » **Regression testing:** Testing the product increment with previous product increments to ensure that previous functionality continues to work
- » **Vulnerability or penetration testing:** Security testing to evaluate the product's exposure to internal and external threats
- » **User acceptance testing:** Validating that the new functionality satisfies the acceptance criteria
- » **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 and team development techniques

Peer review and pair programming are techniques teams use to build product increments. *Peer review* simply means that development team members review one another's work. *Pair programming* means that two people work together, with one person driving (the pilot) and one observing from behind (the navigator). Both practices improve product quality, build or expand team member capability, and reduce single-point-of-failure exposure.

A newer trend gaining momentum is mob programming. *Mob programming* is an approach for product development in which the entire team works on the same thing, at the same time, in the same space, and at the same computer. The entire team continuously collaborates at a single computer to deliver a single work item at a time. Customers are often invited to participate with the team as well. Mob

programming extends the benefits of pair programming from two people to the entire team.

Benefits of mob programming include a broader technical understanding of the product, a faster resolution of communication and decision-making problems, preventing the need to do more than is barely sufficient, reduced technical debt, reduced thrashing of the team and team members, and reduced work in progress.

However the team chooses to review each other's work, collocation helps make the review easy and informal — you can turn to the person next to you and ask him or her to take a quick look at what you just completed. 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 discussed later in this chapter. 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 (accepts or rejects) user stories throughout each day as the development team completes them.

As discussed in Chapter 10, 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 11-6 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 11-6:
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 its acceptance criteria as well as 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

A major part of the scrum master's role is managing and helping 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, software, or access 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 scrum teams. Centralized tools, budget restrictions, and standardized processes that don't align with agile processes can all cause issues for scrum 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 5.

Beyond the primary focus of creating shippable functionality, other things happen during the day. Many of these tasks fall to the scrum master. Table 11-1 shows potential roadblocks and the action that the scrum master can take to remove the impediments.

TABLE 11-1 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 functionality.	Do some research to estimate the cost of the software, prepare a summary with 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 sprint. 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 (stakeholder, customer, or subject matter expert) 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 progressing.
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.

Information Radiators

Each day, the team uses information radiators to broadcast important information to not only itself but also to stakeholders. An *information radiator* is a poster, task board, list, or any artifact that can be viewed on-demand. Information radiators such as a sprint backlog or a task board reduce questions such as, “What’s the status of a story?” or “Is the team on track to meet the sprint goal?” Most information radiators are posted in the team’s physical workspace or, if the team is collaborating with other teams, in a common collaboration or meeting area. If teams are using digital collaboration tools, these information radiators are clearly made transparent through obvious links and easy access.



WARNING

Low-fidelity tactile information radiators are our favorite because they’re in your face and can’t be ignored, even unintentionally. They are referenced more frequently than digital tools, which you have to click or search to find. One way that a scrum master creates an environment for team success is by ensuring transparency of useful artifacts through information radiators.

Information radiators that teams find helpful include the following:

- » **Product vision statement and product roadmap:** Provides constant visibility and clarity on the strategic product direction. See Chapter 9.
- » **Product canvas:** Visualizes personas, needs, objectives, and other considerations established at the beginning of product development, and can be updated as the scrum team learns more about the customer and market-place. See Chapter 4.
- » **Product backlog:** Helps scrum team members and stakeholders visualize product capability and what priorities are coming up next. See Chapter 9.
- » **Sprint backlog:** Displays the scope of the sprint and the status of each task.
- » **Task board:** Displays the status of each user story in the sprint. See an example of a task board earlier in this chapter.
- » **Team working agreement:** Reminds the team of the behaviors it agreed to uphold as it works together. The agreement can be updated during sprint retrospectives and other team discussions. See Chapter 8.
- » **Release and sprint burndown charts with goals:** Visualizes daily the progress and trends of each iteration towards goals. See Chapter 10.
- » **Definition of done:** Reminds the team what shippability means and the work each user story requires. Updated during retrospectives and as the team's abilities evolve. See Chapters 2, 10, 12, and 17.
- » **Personas:** Reminds the team through visualization of who their customers are as the team performs their work. See Chapter 4.
- » **Agile Manifesto, Agile Principles, and scrum values:** Reminds the team of the guiding values and principles it is trying to enable, and are referred to frequently by scrum masters and agile mentors as they coach the team throughout the day. See Chapters 2 and 7.

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 the progress of tasks 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 tool



TIP

that the scrum team uses for tracking progress, the sprint backlog data may automatically update the sprint burndown chart as well.

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 and effort remains, which informs the team as to whether it 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. Also, we follow the rule that it should take a development team less than one minute to update enterprise status reporting. If it's taking longer than that, you have the wrong tool. Our free sprint backlog template mentioned earlier in the chapter can help with this. To download it, go to www.dummies.com/go/agileprojectmanagementfd3e.

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 should review the sprint backlog or task board for any risks or impediments 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.

IN THIS CHAPTER

- » Showcasing work and collecting feedback
- » Reviewing the sprint and improving processes

Chapter **12**

Showcasing Work, Inspecting, and Adapting

At the end of each sprint, the scrum team gets a chance to put the results of its valuable work on display in the sprint review. The sprint review is where the product owner and the development team demonstrate to the stakeholders the sprint's completed and potentially shippable functionality. In the sprint retrospective, the scrum team (the product owner, development team, and scrum master) review how the sprint went and determine possible improvement opportunities for the next sprint. Underpinning both events is the agile concept of inspect and adapt, which Chapter 9 explains.

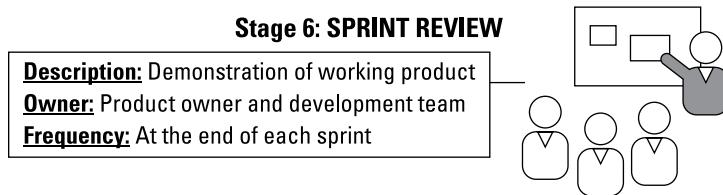
In this chapter, you discover how to conduct a sprint review and a sprint retrospective.

The Sprint Review

The *sprint review* is a meeting to review and demonstrate the shippable and valuable functionality that the development team completed during the sprint, and for the product owner to gather feedback and update the product backlog accordingly. The sprint review is open to anyone interested in reviewing the sprint's accomplishments. This means all stakeholders get a chance to see progress on the product and provide feedback.

The sprint review is stage 6 in the Roadmap to Value. Figure 12-1 shows how the sprint review fits into agile product development.

FIGURE 12-1:
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, for scrum teams, the essence of showcasing is informality. The meeting needs to be prepared and organized but doesn't require 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 and shippable 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 knows exactly what the development team completed in the sprint because he or she was working alongside them to accept or reject the items completed as valuable and shippable. The development team needs to be ready to demonstrate completed, shippable functionality.

The time needed to prepare for sprint review should be minimal — usually no more than 20 minutes — just enough 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, or *shipped*, to the market 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 11.

For the development team to demonstrate the functionality in the sprint review, it must be complete according to the definition of done. In other words, the product increment 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 product meets these standards as well as the user stories' acceptance criteria. 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 prepare you to confidently start the sprint review meeting.

The sprint review meeting

Sprint review meetings have three activities: Demonstrate and showcase the scrum team's finished work, allow stakeholders to provide feedback on that work, and make product adaptations based on reality and stakeholder feedback. Figure 12-2 shows the different loops of feedback a scrum team receives about a product.

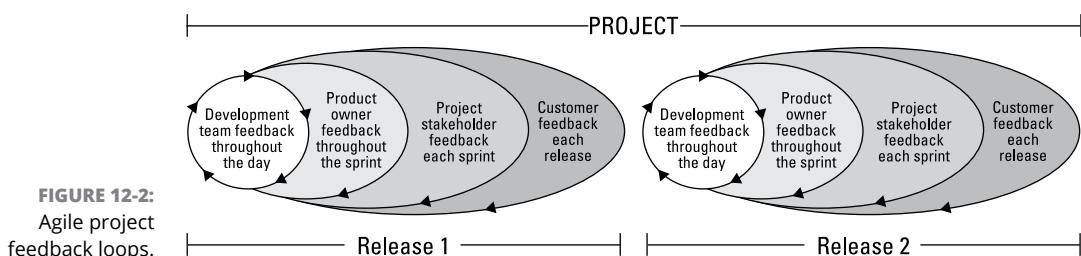


FIGURE 12-2:
Agile project
feedback loops.

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 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, end-customers 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 for a sprint that starts on a Monday. One of the rules of scrum is to spend no more than four hours in sprint review for a one-month sprint, so you usually spend no more than one hour in a sprint review meeting for every week of the sprint, as shown in Figure 12-3.

If my sprint is this long...	My sprint review meeting should last no more than...
One week	One hour
Two weeks	Two hours
Three weeks	Three hours
Four weeks	Four hours

FIGURE 12-3:
Ratio of sprint
review meeting to
sprint length.

Here are some guidelines for your sprint review meeting:

- » No 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 product may attend. The product 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.
- » The development team demonstrates the working product increment *completed* during the sprint. Typically, the development team showcases new features or architecture. When negative or critical feedback is given, resist the temptation to become defensive because it will discourage stakeholder feedback.
- » The demonstration should be on equipment and environments 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 from a simulator on a laptop.
- » The stakeholders can ask questions and provide feedback on the demonstrated product increment.
- » Do not use 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 work for the scrum team in future sprints, catching up to what the stakeholders think already exists. Build trust by setting accurate expectations.
- » Sprint reviews are excellent opportunities to reflect with transparency on the remaining budget to help the product owner evaluate the value of the remaining backlog.



TIP

The formula $AC+OC>V$, which is discussed in Chapter 15, is helpful here for deciding when to stop or shift development. When the actual cost (AC) plus the opportunity cost (OC) of the future development is greater than the value (V), stop or shift development. Sprint reviews are great opportunities for concluding with stakeholders that future investment in the feature or product development should end.

- » The product owner can lead a discussion about what is coming next based on the features just presented and new items 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 will be presented and has agreed that they are complete. If the product owner does not accept a user story that the development team worked on, the story doesn't get demonstrated in the sprint review. As the product is iteratively built sprint after sprint, the sprint review is critical for ensuring alignment with the stakeholders and the product vision. Every sprint, the team should be obsessed with solving customer problems.

The sprint review meeting is valuable for the development team because it can show its work to stakeholders and customers and get its efforts acknowledged. The meeting contributes to development team morale, keeping the team motivated to accomplishing the product vision, solving customer problems, and achieving the desired business outcomes. Stakeholder confidence and trust in the development team increases as they hear the team use business language during the product demonstration. (Another benefit of having stable teams is that they retain hard-earned knowledge about the customer and the business.)



TIP

Inspection is a valuable tool. The largest number of world records are broken at the Olympics. Why? Because millions of people from all over the world are watching. Sprint reviews provide a similar, albeit much smaller, stage for scrum teams. Accountability is higher because of the frequently exposed transparency of their work.

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. Capturing feedback publicly on a whiteboard, for example, validates that the feedback was given and received as intended. The transparency also prevents duplicates.

Because the sprint goal was selected based on the team's assumptions about what the customer wanted, the sprint review offers the team the opportunity to validate its assumptions with stakeholders and, even better, customers.

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 stories could be new features or changes to existing functionality. Both are welcome.



TIP

In the first few sprint reviews, the scrum master may need to remind stakeholders about agile principles and practices. Some people hear the word *demonstration* and immediately expect fancy slides and printouts. The scrum master can shield the scrum team by managing these expectations and helping stakeholders 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 to the product backlog any stories scheduled for the current sprint but not completed, and reorders those items 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’s time for the sprint retrospective. 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.

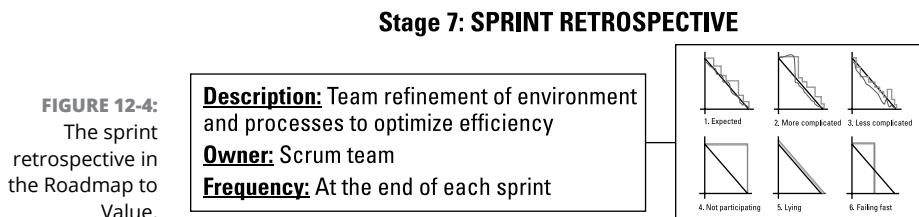
Having just completed the sprint review, the scrum team will come into the retrospective ready to inspect its processes and will have ideas for adaptation.

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’s ability to inspect and adapt in a self-organizing way.

The team might invite others with whom they regularly interact (such as stakeholders) to participate in the sprint retrospective, but these invitations are generally an exception.

The sprint retrospective is stage 7 in the Roadmap to Value. Figure 12-4 shows how the sprint retrospective fits into agile product development.



The goal of the sprint retrospective is to continuously improve your processes, environment, collaboration, skillsets, practices, and tools. Improving and customizing how people work together according to the needs of your scrum team increases scrum team morale, improves effectiveness in achieving desired outcomes, and increases *velocity* — work output. (See Chapter 15 for details on velocity.)

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 to work early and leave early, so they could spend summer afternoons with their families. Members of 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, effectiveness, and velocity.

Use the information you learn in the retrospective to review and revise your work processes and make your next sprint better.



REMEMBER

Agile approaches — particularly scrum — quickly reveal product development problems. 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. 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.

STOPPING THE LINE

Taiichi Ohno, who built the Toyota Production System in the 1950s and '60s — the beginning of lean manufacturing — decentralized assembly-line management to empower the 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 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 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 latter teams declared this a victory at first. However, it didn't take long until the former 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.

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 retrospectives

For the first sprint retrospective, everyone on the scrum team should think about a few key things and be ready to discuss them. What went well during the sprint and what should we keep doing more of? What would we 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 11, we mention saving sprint backlogs from prior sprints; this is one instance 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 will go into the sprint retrospective ready to have a useful and actionable conversation.

The retrospective meeting

The 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 *because . . .*, the conversation is moving away from action and toward rationale.

One of the rules of scrum is to spend no more than three hours in sprint retrospective for a one-month sprint. So you usually spend no more than 45 minutes in a sprint retrospective meeting for every week of the sprint. Figure 12-5 shows a quick reference of this timetable.

If my sprint is this long...	My sprint retrospective meeting should last no more than...
One week	45 minutes
Two weeks	1.5 hours
Three weeks	2.25 hours
Four weeks	3 hours

FIGURE 12-5:
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 examples of topics for inspection:

- » **Results:** Compare the amount of work planned with what the development team completed. Review the sprint or release burndown charts and what they tell the team about how it's working.
- » **People:** Discuss team composition and alignment.
- » **Relationships:** Talk about communication, collaboration, and how the team works together.

- » **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 while maintaining a sustainable pace? Remember Principle 8, "Agile processes promote sustainable development." Perhaps there are opportunities to "maximize the amount of work not done" (Principle 10) or to work smarter?

If the team works in one-week sprints, it will have nearly 52 opportunities each year to hold a retrospective. To maintain engagement in each retrospective, many retrospective formats exist, and it's helpful for the team to have variety and to have these discussions in a structured format. Esther Derby and Diana Larsen, authors of *Agile Retrospectives: Making Good Teams Great* (Pragmatic Bookshelf, 2006), offer a great framework for sprint retrospectives that keep the team focused on discussions that will lead to real improvement:

1. Set the stage.

Establishing the goals and scope for the retrospective up front 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 data 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 want to put into place. Decide on specific actions you can take to make the ideas 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 facilitates these discussions to guide them towards the desired outcome and keeps the meeting within its allotted time.



TIP

Any action item coming out of a sprint retrospective should be added to the product backlog. All work the scrum team will do to achieve the product vision, such as features, technical debt, overhead, and improvements, should be added and prioritized in the product backlog. Scrum teams should agree to include at least one improvement item from a previous retrospective in each sprint to continuously improve how they go about the work of delivering potentially shippable functionality every sprint.

Be sure to use the results from your sprint retrospectives to inspect and adapt every sprint throughout your product development, not just when it is convenient.

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. Don't forget to add action items to the product backlog as a reminder to implement them during an upcoming sprint.

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. High-performing teams have learned how to convert retrospectives into velocity acceleration.

Agility Management

IN THIS PART . . .

Incrementally pursue creating value (outcomes) rather than meeting specifications (outputs).

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.

IN THIS CHAPTER

- » Learning how agile portfolio management is different
- » Discovering ways to make portfolio investment decisions
- » Learning how to manage an agile product portfolio

Chapter **13**

Managing a Portfolio: Pursuing Value over Requirements

Organizational leaders today face the daunting challenge of delivering value faster than ever before. With limited resources and funding to keep up with rapidly changing marketplaces, deciding which product investment opportunities will maximize an organization's return is not easy. In this chapter, we share various agile approaches and techniques used to manage a product portfolio. Portfolio management helps organizations to pursue value (outcomes) over meeting requirements (outputs).

Like product development, agile portfolio management is based on the Agile Manifesto values and principles and yields the following results:

- » Highest business value delivered first
- » More frequent delivery of value
- » Lowered cost of pivoting (reprioritization)
- » Higher transparency

- » Timely data from shorter feedback loops
- » Ease in managing by exception due to team and product transparency
- » Small, incremental continuous improvement
- » Sustainable productivity
- » Improved morale due to improved focus and sustainable pace
- » Not only fulfilled requirements, but also maximized value

Understanding the Differences in Agile Portfolio Management

Because scrum teams continually focus on the highest value and risk items first, agile product development efforts often run out of value before they run out of time or money. The same can be said of a portfolio of products. In fact, many of the same principles for maximizing value from a single product can be applied to a portfolio of products.

Agile portfolio management is the art and science of selecting and overseeing a group of product investments that meet an organization's long-term objectives and risk tolerance. Managing a portfolio of products requires the ability to weigh strengths and weaknesses, opportunities, and threats across the full spectrum of opportunities. The choices involve trade-offs: short term versus long term, expanding market segments versus contracting ones, domestic versus international, and growth versus existing product investment.

Weighing the various trade-offs truly is an art and a science. Product owners and portfolio leaders evaluate strengths, weaknesses, opportunities, and threats (SWOT). Effective portfolio management enables maximization of value for customers and stakeholders.



A SWOT analysis can be conducted on anything the team wants, such as its product, company, team, or portfolio. The sky is the limit. After determining the topic of their conversation, the team brainstorms the strengths, weaknesses, opportunities, and threats. The outcome of the analysis is a concise summary of the most important factors to be considered by the team, which may guide the product portfolio, backlog, vision, and roadmap.

All agile principles can be helpful for managing organizational product portfolios:

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.

As a portfolio of products is evaluated from a macro level, key decisions must be made about which product investment opportunities should be pursued in general, which should be pursued next, and which might be pursued later. A careful balance must also be set to ensure that the organizational and team capacity is healthy and thriving. Overburdening leads to reduced innovation, tiredness, and acceptance of the status quo.

Should we invest?

As with financial investments, capital budgeting uses a key metric to estimate the profitability of potential investments: the internal rate of return (IRR). IRR considers the cost of the initial investment against the net present value of the future stream of revenue or cost savings. The higher the IRR, the more confident a portfolio leader can be that the product investment will be profitable.

These same financial return principles can be applied to product portfolio management. The cost of the initial product, including development, labor, licensing, and maintenance, is compared to the future stream of annualized revenue or cost savings. If the internal rate of return exceeds the cost of capital and the initial investment cost, it might be a worthwhile investment. Other costs to consider are the balance between CapEx (capital expenditures) and OpEx (operational expenditures). The more expenses that can be capitalized, the more profitable an investment.



TECHNICAL STUFF

Capital expenditures, or CapEx, are funds used by a company to acquire, upgrade, and maintain physical assets such as property, buildings, an industrial plant, technology, and equipment. CapEx is often used to undertake new projects or investments by the firm. An operating expense — also known as an operating expenditure, operational expense, or operational expenditure (OpEx) — is an ongoing cost for running a product, business, or system.

Understanding the internal rate of return and balance between CapEx and OpEx helps answer the question, “Should we invest?” Following are other factors to consider in forecasting financial product return:

- » **Value and risk:** Portfolio prioritization is a factor of value and risk. Value, because the portfolio must meet the customer's needs. Risk, to enable failing early and failing cheap, giving you the longest runway for finding a solution when the system is the simplest and the most financial resources are available.
- » **Short-term versus long-term:** Trade-offs of short-term gains for long-term value.
- » **Product mix:** Balance of products to diversify and take advantage of product lifecycles of both new and sunsetting products (discovery to market).

We discuss each in more detail next.

Factors for forecasting product investment returns

Several factors, such as value and risk prioritization, short-term versus long-term, and balancing the product mix, should be considered when forecasting product investment returns. We discuss each of these in the next section.

Value and risk prioritization

Agile product teams are encouraged to focus on one thing, do it well to completion, and then move as a team to the next most important thing. The team works on one item at a time to avoid the significant cost delays of context switching.

Prioritization across a portfolio is similar. The portfolio may have a backlog of investment opportunities, but wise portfolio leaders reduce work in progress and focus on one opportunity at a time. They evaluate the desired outcomes of their focus, and once acceptable, move to the next opportunity.

They prioritize the portfolio according to value and risk. Value criteria could include return on investment and internal rate of return, market share, revenue, cost savings, corporate image, product enhancements, maintenance, security, regulatory compliance, and more. Risk criteria may include the risk of users adopting a new product, of using a technology unfamiliar to the organization, of the organization being unable to deliver the value in the time frame needed, and more. Portfolio prioritization is a function of value and risk. Learn more about using value and risk for prioritization in Chapter 9.

Figure 13-1 shows a helpful value and risk matrix tool that product portfolio leaders use to evaluate various product investment opportunities. Products in the highest value, highest risk quadrant should be attempted first while products with the lowest value, highest risk should be avoided.

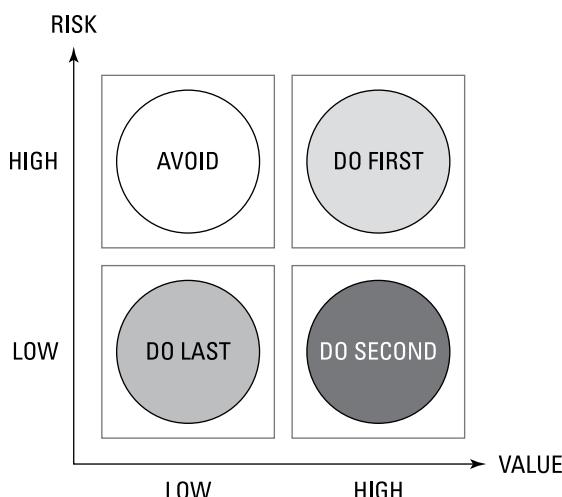


FIGURE 13-1:
Risk versus value quadrant.

GAINING ORGANIZATIONAL AND PORTFOLIO PRIORITY ALIGNMENT

A healthcare client of ours struggled with annual planning due to the traditional project management approach used by the organization. Capital was reserved and forecasts were continually updated throughout the year so allocation adjustments could be made. Managers watched the CapEx and OpEx targets like hawks. Progress was difficult to make with so many active and dependent pieces of work going all at once. People with specialized skills were required to work on multiple projects at the same time.

In planning the new year, a professional agile mentor was brought in to help. A workshop was scheduled to help them work through the budgeting and productivity challenges. The mentor walked the group through an approach that helped them to begin by agreeing to criteria for evaluating risk and value.

Each leader then filled out 3x5 cards with the title of their planned product investment opportunity and placed it on a risk-value matrix on the wall. In the value label field, they recorded their estimated return on investment (\$500K, \$300K, \$100K, and so on) and defended their ROI estimate as each card was discussed.

After all the cards were placed on the wall, the first acknowledgment they made was that there were too many investment opportunities — their plan was unrealistic. The second observation was that they could prioritize their portfolio by understanding the value, risk, and rough order estimate. The subset of the most valuable opportunities emerged by priority and, more importantly, had organizational alignment.

The client then laid the best opportunities in a spectrum across the wall in prioritized order, starting with the most valuable/most risky opportunities on the left and the least valuable/least risky opportunities on the right. In essence, they created their backlog of product portfolio investment opportunities.

The results were significantly improved. Highest priority opportunities were either successful earlier or the high risk, low value opportunities were quickly removed from the list. The organization's teams could pull from a prioritized backlog of opportunities when they became available, allowing more and more of the organization's talent to focus on delivering the work that mattered most at a pace in their available capacity.

A prioritized and ordered portfolio of product opportunities enables teams to pull from the backlog of investment opportunities. After the first available team pulls from the backlog, the next team checks to see if the first team needs help; if not, it pulls the next most valuable product investment opportunity, and so forth. This approach ensures that the highest valuable product investment opportunities

receive the organization's attention first. It also helps to keep the work in progress (WIP) low while moving at the pace of the organization's capability.

Short-term versus long-term decisions

Portfolio leaders, in partnership with product owners, make short-term versus long-term product portfolio decisions. The question they often ask is, "Will my short-term product investment opportunity be thrown away?" Or, "Will my short-term investment create a path to accomplish the longer-term strategic vision?"

SHORT-TERM PAYMENT OF ORGANIZATIONAL TECHNICAL DEBT YIELDS LONG-TERM RESULTS

The short-term decision to invest in paying down organizational technical debt weighed heavily on the minds of the healthcare organization's newly formed agile transition teams (ATT) as they took seriously their responsibility to support the agile transformation. (See Chapter 18 to learn more about agile transition teams.) As the pilot scrum teams started working, the lack of test automation became exposed. Thousands of test cases that could have been automated were manually used day-in and day-out by everyone. Each new day of created functionality caused more manual test cases to be created, making the pile of manually testing debt even higher.

After consulting with the scrum teams, the ATT decided that the best approach for addressing the debt was to focus a new scrum team on the problem.

The first item on the new scrum team's backlog was to build a testing framework that could be used by all teams. Each pilot team embedded the new testing framework in its definition of done. The framework allowed all the teams to automate their own tests, adding them to the shared pool of tests. While the scrum teams automated the tests on their new sprint's work, the newly formed test automation scrum team tackled the backlog of test automation debt using the new framework. Test case by test case, the debt was paid down.

The product's quality as well as the teams' productivity improved. Changes were much easier to introduce and the teams shifted priorities effortlessly. Manual testers gained new skills in building test automation, which opened doors for learning other skills.

In the long-term, the velocity, or relative pace, of development across all the teams increased due to the investment in automation. The short-term payment of technical debt yielded long-term results.

Scrum teams frequently ask, “What should we do next?” Decisions must be made to either fix something or build or implement something new. A balance between proactive work and reactive work as well as the decision to pay down technical debt or invest in automation or platform upgrades for faster product development must be decided.

To answer these short-term versus long-term decisions, portfolio leaders work closely with product owners, stakeholders, and development teams. The strong communication cycles and quick feedback loops inherent in agile approaches inform their decisions. As they interact with stakeholders and product owners, they learn about customer needs and opportunities. The product owners and stakeholders then quickly build product vision statements and roadmaps with development teams to gain funding support. Release plans and sprint plans follow. Initial sprints give early information about the viability of the opportunity. With minimal investment, information is revealed to inform their short-term versus long-term decision.

Balancing the product mix

Product mix, also known as *product assortment*, refers to the total number of product lines an organization offers to its customers. For example, think of the different products sold by Nike (shoes, socks, pants, shirts, sweatshirts, team gear, sports equipment, and so on). The four dimensions to a company’s product mix include width, length, depth, and consistency:

- » **Width:** The number of product lines that an organization offers or the variety offered, for example, the number of product lines from shoes to other clothing or team gear.
- » **Length:** The number of products in the company or a given product line, for example, the number of different running shoe types.
- » **Depth:** The total number of variations for each product, such as the various sizes of shoes, styles, or colors.
- » **Consistency:** The connection between products in the product line and the way they reach the consumer. It describes how closely product lines are related to each other — in terms of use, manufacturing, and distribution. For example, two lines of shoes may be used, manufactured, and distributed in the same way so that their product lines are consistent.



TIP

The product mix is important in determining the image of your business and brand, because it helps you maintain consistency in the eyes of your target market.

Various strategies can be employed to balance the product mix. Small companies usually start with a product mix limited in width, depth, and length and have a high level of consistency. Over time, the organization may want to differentiate products or acquire new ones to enter new markets. They may also add to their lines similar products that are of higher or lower quality to offer different choices and price points. They stretch the product line — upward to add more expensive products and downward to add lesser quality, lower priced items.

A well-balanced marketing mix in a product portfolio is important for a number of reasons. It helps portfolio leaders to maximize the value of their investment. It defines the steps necessary to make a profit by differentiating the products and to take advantage of new markets, customer segments, and price points. Lastly, it sets the course for product marketing.

Managing Agile Product Portfolios

The many moving parts involved in managing an agile product portfolio can make the process complex. Effective portfolios focus on setting realistic and strategic priorities, and reducing work in progress so portfolio items can get completed as quickly as possible. They adhere closely to Principle 10: Simplicity, the art of maximizing the amount of work not done — is essential.

Brent Barton, in his article “The 5 Simple Rules of Agile Portfolio Management,” describes the challenges when managing and planning a product portfolio. He concluded that planning new capabilities for multiple products in an organizational silo was like predicting the weather in micro-climates with a less than 50 percent accuracy. The accuracy was so low because the micro-climate meteorologists weren’t looking (or couldn’t look) beyond their own domain to account for how the bigger system affected weather patterns. In other words, product portfolio planning without the proper perspective for reasonably matching capacity to demand is nearly impossible.

For this reason, Barton defined five simple rules to reduce the complexity of agile product portfolio management:

- » **All work is forced ranked.** Organizations that make everything the highest priority cannot articulate real priorities. Force ranking a portfolio of investment opportunities leads to clarity and focus. Similar to answering the “Should we invest?” question, portfolios are best prioritized when they consider value to the customer as well as business risk, technical risk, or both. Tackle the most valuable and most risky opportunities first.



REMEMBER

In his book *Essentialism: The Disciplined Pursuit of Less* (Virgin Books), Greg McKeown said:

The word priority came into the English language in the 1400s. It was singular. Only in the 1900s did we pluralize the term and start talking about "priorities." Illogically we reasoned that by changing the word we could bend reality.

- **Operate on good enough data.** It is unrealistic to expect to have perfectly detailed data for all portfolio decisions when they need to be made. Certain levels of detail in one area do not necessitate expensive-to-obtain detail in other areas. As with all empirical control processes, start with what you know, and then inspect and adapt based on what is learned.
- **Near-term capacity is fixed.** Make investment opportunity decisions based on existing organizational capacity. Adding complexity by forecasting team or skill counts based on the hope influencing portfolio management outcomes is problematic. Forecasting and onboarding new teams in a timely manner is more challenging than leaders want to acknowledge.
- **Each unique value-based delivery capability has a portfolio.** Simplify and decompose the work into the smallest, most valuable increments as possible. Enable teams in the portfolio to become highly aligned and highly autonomous. Connect their work to strategy through progressive elaboration.
- **Each portfolio has one intake system.** Strategic decisions in a portfolio should have full visibility into the entire scope of work required to innovate, build, release, evolve, support, and sunset technology.

These five simple rules for agile product portfolio management help reduce complexity, enable organizational focus, and reward action and speed over analysis paralysis. Like all guidelines, they serve as a good starting point but will need to be adjusted upon inspection.

Other factors to consider for effectively prioritizing a product portfolio follow:

» **Visualize the portfolio.** A map to your destination is easier to follow than verbal or written directions. Similarly, visualization enables portfolios of product teams to collaborate more effectively. In a self-organizing environment, full transparency is essential for inspecting and adapting.

Ensure that everyone can see how your product portfolio aligns to your corporate vision, objectives (desired outcomes), success criteria, and strategy. Your roadmap of products and capabilities needs to be visible and supportive of the decision-making process. Visualization helps to inform decisions that can be made at the last responsible moment.

Figure 13-2 shows an example of a prioritized portfolio backlog of opportunities aligned with the strategic vision stack, ranked by value, risk, and estimate.

- » **Optimize talent allocation.** Ensure that talented scrum teams are optimized for creating maximum benefit.
- » **Evaluate product performance.** Ensure that customer driven metrics are available for adapting to changing market conditions and customer demands.
- » **Determine prospective value.** Determine new products to design and develop as well as search for new ideas. Monitor your business and your competitors, getting feedback from customers and anticipating clients' future needs. New technologies, laws, or regulations may also influence prospective value.
- » **Search possible endeavors.** Search for and invest in understanding possible endeavors. Options include guessing at a product's market potential, doing a case study, or running a focus group.
- » **Commence endeavors.** New products or features have to be developed by the team. Initial experimentation can help you see the market potential of the product.

Portfolio Backlog of Opportunities

Organizational Strategic Vision:

To become the premiere financial services agency most admired for its people.



FIGURE 13-2:
Prioritized
portfolio backlog
of investment
opportunities.

- » **Incrementally fund products.** Every potential endeavor requires funding, including the primary funding for new teams for their inception or development efforts and the budget for ongoing construction, transition, and operations after the product launch. Instead of allocating funding for the entire development effort, consider incrementally funding instead. For example, allocate 90 days' worth of funding and see what kind of value can be created in that time. If successful, fund another 90 days. This funding will also require constant monitoring by the product owner and sponsors to ensure that the money is spent wisely.
- » **Engage vendors.** Supplier or vendor management is a key aspect of agile portfolio management. This task entails procuring or awarding contracts, narrowing down and identifying possible vendors, overseeing ongoing projects, and closing a contract. Ensuring that vendors enable agile principles is essential because they'll need to adapt with your changing priorities and customer needs. Many times, you can use an incremental funding model with vendors to more effectively manage portfolios. Enable product owners or scrum masters to engage in vendor relationships early.

Table 13-1 lists important considerations for effective portfolio management with suggestions for what to do and what not to do in each situation.

TABLE 13-1

Keys for Effective Agile Portfolio Management

Topic	Do	Don't
Partner with product owners	Empowered product owners are close to their customers and understand their needs and problems. Engaging product owners in portfolio decisions leads to better aligned portfolios.	Don't exclude product owners from contract negotiations, discovery, or portfolio decisions. Product owners are accountable for a product's return on investment.
Prioritize and limit the work to be done	Tackle the single, next most valuable opportunity, do it well, validate that the desired outcomes were achieved, and then move to the next opportunity. Prioritization and limiting work to be done improves speed to market and organizational focus and builds organic cross-functional capability.	Don't enable teams to work on lower value, low risk work at the expense of the higher value, higher risk work. Working on too many things at once will prevent you from working on the higher priority things that matter most to your customer.
Focus on balancing demand and capacity	Avoid overburdening teams and organizations. Overburdening leads to tired and inefficient people who will make more mistakes, which cause delays and are expensive to correct. Even worse, their ability to innovate will be constrained.	Don't push velocity expectations onto a team. Scrum teams are more effective when they pull work into releases and sprints given their capacity constraints.

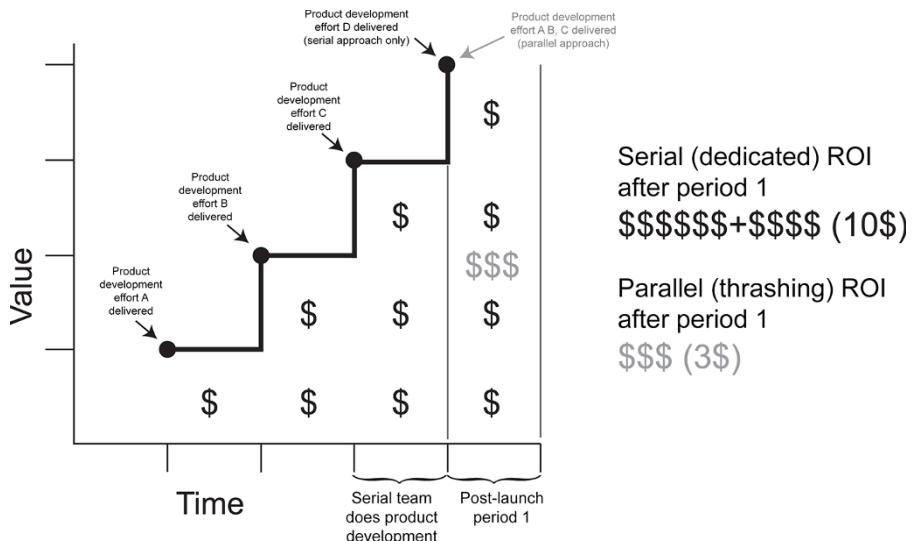
Topic	Do	Don't
Maximize both internal and external value	Focus on creating value for your customers as well as keeping a broader perspective of doing what's best for your organization. Agile product portfolio management is more about improving your organization for the future and less about completing projects. It's about directing funds where they'll have the most effect.	Don't focus solely on internal or external value. History shows that internally focused businesses who lose sight of their customers or who neglect to take care of their workforce cease to exist.
Re-prioritize frequently	As market conditions and customer demands change, inspect and adapt. Continuously prioritize your product investment opportunities.	Don't become irrelevant by not adapting to change.
Deliver value in smaller chunks	When planning your portfolio, focus on how small and simple you can make the value increments that you'll release to the market. Doing so not only creates a faster cashflow and return on investment but also reduces complexities and risk.	Don't place value on a shelf, like inventory. When a barely sufficient amount of value is created, release it.
Avoid parallel product work	Product portfolio leaders understand that organizations get less done when they work on several products at the same time. Prioritize, and then select the next most valuable opportunity and finish it. Next, pull the next most valuable investment opportunity only when capacity and capability are available.	Don't allow parallel work. Simplify product development by reducing work in process, which has no value.

In Tom Demarco's book *Slack: Getting Past Burn-out, Busywork, and the Myth of Total Efficiency* (Currency), the anti-pattern of parallel product work is further explained. Tom suggests that people and teams are not fungible (not exchangeable or replaceable). People or teams required to support multiple product development efforts in parallel pay a high cost in context switching. According to the American Psychology Association, the cost can be as high as 40 percent of someone's productive time.

A study performed by Gloria Mark from the University of California, Irvine and Daniela Gudith and Ulrich Klocke from Humboldt University in Germany added that people interrupted in their work also experience higher workloads, more stress, higher frustration, and more time pressure and effort. The same principles apply to parallel, interrupted product development.

Figure 13-3 illustrates the difference between running serial product development efforts (dedicated teams) and parallel efforts (thrashing teams).

In this example, a portfolio has three projects, each running in succession (serially). Assume that one unit of value can be produced in one unit of time and each product development effort when complete produces one unit of value (\$).



Thrashing the team, or causing lack of focus by multitasking or context switching, results in at least 30 percent (as much as 40 percent) more time to finish each effort — 33 percent for this scenario. Over the three product development efforts (99 percent), thrashing the teams around is roughly the equivalent of an entire effort's length.

Going one additional unit of time past deployment of the serial efforts, the parallel efforts finally finish and return \$\$\$ (3\$ — one \$ for each effort deployed) at the end of that period. The serial product development efforts return \$\$\$\$\$\$+\$\$\$\$ — 10\$, which is more than three times the return on investment (ROI) compared to parallel development.



REMEMBER

Stop thrashing. Run one project at a time through one team at a time. When you dedicate teams, everyone gets value delivered earlier, as well as an earlier and overall higher ROI. It simply doesn't make sense to overload your portfolio and trash your teams on more than one thing at a time.

Should we continue investing?

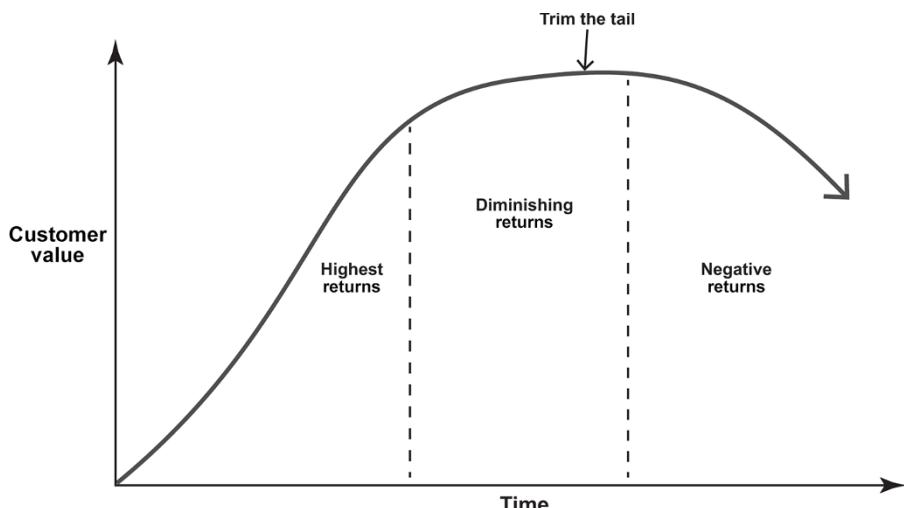
The question of whether you should continue investing is one that all portfolio leaders in collaboration with product owners must answer. Many factors must be considered for making the decision. Leaders and product owners consider “When is the right time to sunset a product?” or “Have we reached the point of diminishing returns, where the investment return will not be as great as other product investments?”



TECHNICAL
STUFF

The *law of diminishing returns*, also referred to as the law of diminishing marginal returns, states that in a production process, as one input variable is increased, there will be a point at which the marginal per unit output will start to decrease, holding all other factors constant.

Figure 13-4 shows how to apply the law of diminishing returns to a product portfolio. During the early stage of the portfolio, knowledge value is earned, followed by a steep increase in customer value (or outcomes) as the new capabilities are made available to customers. (The “highest returns” section is the most productive section of the curve in the figure.) It pays to invest more time and effort during this stage. Next is the “diminishing returns” section, when each added input leads to a decreasing rate of output. It’s best to stop somewhere within this phase and “trim the tail.” In other words, more valuable opportunities should now be considered. The last stage, “negative returns,” is the one to avoid. Not only do you not get a return for your effort, but you decrease your overall output.



When the point of diminishing returns is met, it’s time to stop or reduce investing in a product. In other words, it’s time to shift to the next most valuable investment opportunity using the $V < AC + OC$ equation discussed in Chapter 15.

Inspecting and adapting to the next opportunity

After you’ve achieved the accomplishment of getting to done, what’s next? Maintaining a refined portfolio backlog of investment opportunities makes this

decision easier. This section describes some keys for continuously prioritizing and shifting to the next most valuable investment in your product portfolio.

Shifting to the next most valuable item

Shifting to the next most valuable item is easier to do when a continually prioritized portfolio of investment opportunities is maintained. As with a product backlog, focus on the next portfolio backlog item.

Revising product portfolio investments

If the outcomes from your portfolio investment do not accomplish their intent, it may be time to adjust and adapt. The beauty of an agile product portfolio is its capability to change priorities with minimized disruption. Scrum teams that implement shippable functionality at the end of every sprint have frequent inspection points to determine if enough value has been achieved.



REMEMBER

Agile portfolio management ensures that an organization provides its clients with the best value for their investment while doing what is best for the organization. A good portfolio leader, in partnership with empowered product owners, understands and follows agile principles. They also understand how to align the portfolio to the strategic direction, prioritize, reduce work in progress, and help others to join them in their journey. They pursue value (outcomes) over meeting requirements (outputs).

IN THIS CHAPTER

- » Seeing how agile product development changes scope management
- » Managing scope and scope changes with agile techniques
- » Understanding the different approach that agile practices bring to procurement
- » Managing procurement

Chapter 14

Managing Scope and Procurement

Scope management is part of every product development effort. 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 product development efforts. If you need to look outside your organization for help completing development, you should know how to go about procuring goods and services. You will want to know how to collaborate with vendor teams during the product lifecycle. You should also know something about creating contracts and different cost structures.

In this chapter, you find out how to manage scope 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. 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 the project-funded features of a product.

Traditional project management treats changing requirements as a sign of failure in up-front planning. With agile product development, however, scope is variable so that scrum 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 use agile product development 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.



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 approach supports the manifesto values 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 14-1.

TABLE 14-1

Traditional versus Agile Scope Management

Scope Management with Traditional Approaches	Scope Management with Agile Approaches
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 product development, breaking down and further detailing requirements that are going to be implemented in the immediate future. Requirements are gathered and refined throughout development as the team's knowledge of customer needs and product realities grow.
Organizations view scope change after the requirements phase is complete as a failure.	Organizations view change as a positive way to improve a product as development progresses. Changes later in development, when you know more 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 or swaps those requirements to the product backlog.
The cost of change increases over time, while the ability to implement 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 <i>scope 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 during agile product development, 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 10.

The next section addresses how to manage scope with agile product development.

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 team — the scrum team plus the stakeholders — clearly understands the existing product scope in terms of 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 product development. 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 product development

At each stage in development, the scrum team manages scope in different ways. A good way to look at scope management throughout development is by using the Roadmap to Value, first presented in Chapter 9 and shown again in Figure 14-1.

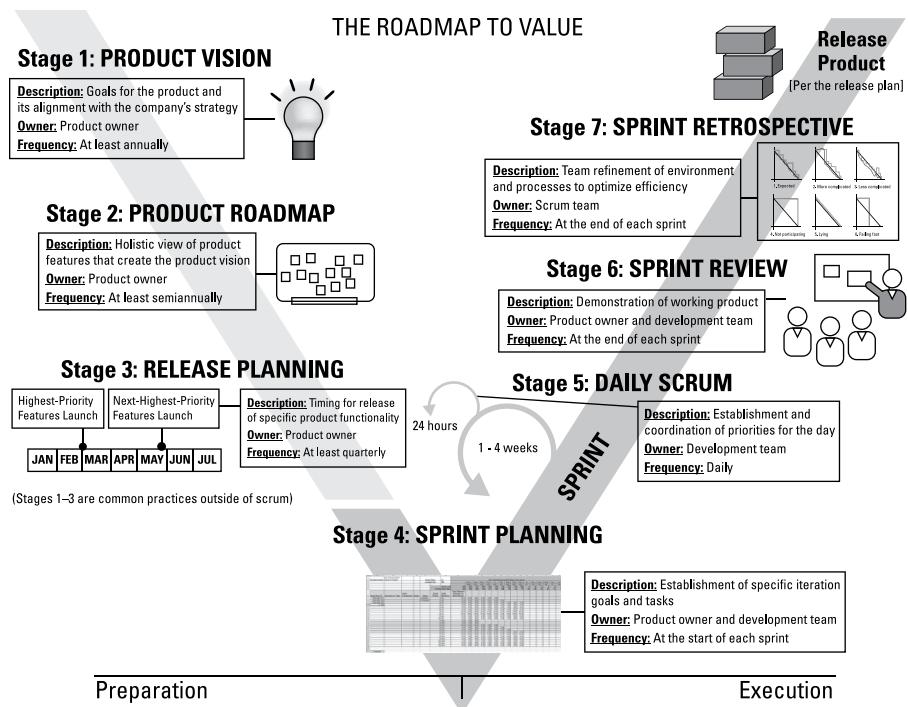


FIGURE 14-1:
The Roadmap to Value.

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 scope. The product owner is responsible for ensuring that all members of the product team know the product vision statement and that everyone on the 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:** In the daily scrum meeting, the scrum team coordinates its focus and progress towards achieving the sprint goal. The meeting can be a launching point for scope change for future sprints.



TIP

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 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 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. 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 product development, 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 product. It may be a good candidate for a separate product.

- (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.

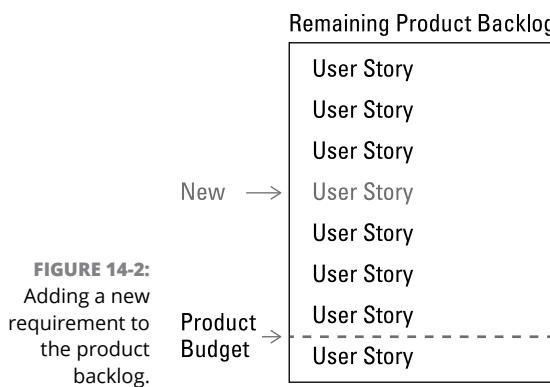
The development team estimates the effort. Find out how to estimate requirements in Chapter 9.

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 product 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 product 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 14-2 shows the addition of a new requirement in 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.



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 remaining scope. Chapter 9 has more information about prioritizing requirements.

Using agile artifacts for scope management

From the vision statement through the product increment, each agile artifact supports 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 9.

Table 14-2 reveals how each agile artifact, including the product backlog, contributes to ongoing scope refinement.

TABLE 14-2 Agile Artifacts and Scope Management Roles

Artifact	Role in Establishing Scope	Role in Scope Change
Vision statement: A definition of the product's end goal. Chapter 9 has more about the vision statement.	Use the vision statement as a benchmark to judge whether features belong in the scope for the product.	When someone introduces new requirements, those requirements must support the product vision statement.
Product roadmap: A holistic view of product features that create the product vision. Chapter 9 has more about the product roadmap.	The product roadmap encompasses the scope of the product. Requirements at a feature level are good for business conversations about what it means to realize the product vision.	Update the product roadmap as requirements arise or change. The product roadmap provides visual communication of the new feature's inclusion in the product.

(continued)

TABLE 14-2 (continued)

Artifact	Role in Establishing Scope	Role in Scope Change
Release plan: A digestible mid-term target focused around a minimum set of marketable features. Chapter 10 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.
Product backlog: A complete list of all known scope for the product. Chapters 9 and 10 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.
Sprint backlog: The product backlog items and tasks in the scope of the current sprint. Chapter 10 has more about the sprint backlog.	The sprint backlog contains the product backlog items 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 agile product development 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 product development.

Chapter 2 explains that the Agile Manifesto values *customer collaboration over contract negotiation*. This sets an important tone for procurement relationships.

Valuing customer collaboration more than contract negotiation doesn't mean that agile development efforts 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. However, the following seem to stand out the most when securing goods and services for product development:

- 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 14-3 highlights the differences between procurement on traditional projects and procurement with agile product development.

TABLE 14-3 Traditional versus Agile Procurement Management

Procurement Management with Traditional Approaches	Procurement Management with Agile Approaches
The project manager and the organization are responsible for procurement activities.	The self-managing development team plays a larger 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 product development 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.	Scrum 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-sprint, the new vendor can immediately start developing requirements for the next sprint, avoiding a long, costly transition.



TECHNICAL STUFF

Both waterfall and scrum 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 approaches, by contrast, are firm in their accountability for end results, defining success as working functionality that achieves the customer's desired outcome.

The next section shows how to manage procurement.

Managing Agile Procurement

This section focuses on how scrum 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 product development effort.

Determining need and selecting a vendor

With agile product development, procurement starts when the development team decides it needs a tool or the services of a third-party to create the product.



REMEMBER

Agile 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 product management areas, including procurement. Find out more about self-managing teams in Chapters 7 and 16.

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. Start mobilizing the procurement at this time.
- » **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:** Product 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.

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 scope against the budget, so the product owner is ultimately responsible for any 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 product development environment and, if so, how much experience the vendor has with agile techniques
- » 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 with agile product development.

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 product development. After you understand these approaches, you see how to create a contract.

Cost structures

When you're procuring services for agile product development, it is important to know the difference between *fixed price*, *fixed time*, *time and materials*, and *not to exceed*. Each approach has its own strengths in an agile setting:

» **Fixed price:** Starts out with a set budget. With a fixed price, 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 development will be able to last 25 weeks. Within those 25 weeks, the vendor creates and releases as much shippable functionality as possible.

» **Fixed time:** 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 a fixed time, you determine costs based on the cost of the vendor's team for the duration of development, along with any additional resource costs, such as hardware or software.

» **Time and materials:** Is more open-ended than fixed priced or fixed time. With time and materials, your work with the vendor lasts until enough product functionality is complete, without regard to total cost. You know the total cost at the end of development, after your stakeholders have determined that the product has enough features to call the product complete.

For example, suppose your development costs \$10,000 a week. After 20 weeks, the stakeholders feel that they have enough valuable product features, so your total cost is \$200,000. If the stakeholders instead deem that they have enough value by the end of 10 weeks, the cost is half that amount, \$100,000.

» **Not to exceed:** Is used when time and materials have a fixed-price cap.

THE FALLACY OF LOW-BALLING THE VENDOR

Trying to bully vendors into providing the lowest possible price is always a lose-lose proposition. 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 client 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 the least about the project. Change orders — and their accompanying cost increases — are inevitable.

A better model is for the vendor and client to collaborate on defining the scope, within fixed cost and schedule constraints, as product development unfolds. Both parties can reap the benefits of what they learn during development, 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.



REMEMBER

Regardless of the cost approach, with agile product development concentrate on completing the highest-value product features first.

Contract creation

After you know the 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* imply that contracts are unnecessary (“customer collaboration over contract negotiation”). Regardless of the size of your company or organization, you should create a contract between your company and your vendor for services. Skipping the contract can leave buyers and sellers open to confusion about expectations, can result in unfinished work, and can even lead to legal problems. Contracts should develop through customer collaboration, implying a non-adversarial approach.

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 agile product development 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 9.
- » **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 11): 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 visualizing progress
 - People whom the vendor will assign to the product, such as the development team
 - Where the vendor will work, such as on-site at your company
 - Whether the vendor will work with a scrum master from the vendor’s organization or from your organization
 - 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 client’s development team and sprints**

This is not a comprehensive list; contract items vary by product 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 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 to work with a vendor.

Working with a vendor

How you work with a vendor on an agile product development effort depends in part on the vendor team's structure. In an ideal situation, vendor teams are fully integrated with the client's organization. The vendor's team members are collocated with the client's scrum team. Vendor team members work as part of the client's development team for as long as necessary.



REMEMBER

Just because the vendor is not part of your organization, it doesn't mean the vendor's team members are not part of the scrum team. Because you want your vendor integrated as part of your development team, scrum teams include vendor team members in all scrum events.

Vendor teams also can be integrated but dislocated. If the vendor can't work on-site at the client's company, it can still be part of the client's scrum team. Chapter 16 has more information on team dynamics.

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 client's scrum team. See Chapters 15 and 19 to find out how to work with more than one scrum team.

If a vendor doesn't use agile product management processes, the vendor's team works separately from the client'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 client'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 product development with dislocated teams" section in Chapter 16 for information about working with non-scrum teams.

Vendors may provide services for a defined amount of time, or for the life of the development effort. After the vendor's work is complete, the contract is closed.

Closing a contract

After a vendor completes work on a contract, the client's scrum master usually has some final tasks to close the contract.

If development finishes normally, according to the contract terms, the scrum master may want to acknowledge the end of the contract in writing. If the contract stipulates time and materials, 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 client's company accounting department after work is complete to ensure that the vendor is paid properly.

If product development finishes before the contract dictates the end (enough value has been delivered to redeploy capital on a new product development effort), 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 could potentially work together again, and a simple, sincere "thank-you" can help maintain a good relationship for future development.

IN THIS CHAPTER

» Understanding what's unique about time management with agile product development

» Recognizing how cost management is different with agile product development

Chapter **15**

Managing Time and Cost

Managing time and controlling costs are key aspects of managing agile product development. 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 and how to increase development speed to lower your product's time and cost.

What's Different about Agile Time Management?

With agile product development, *time* refers to the processes that ensure timely completion and the effective use of time. 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 15-1 shows some of the differences between time management on traditional projects and agile product development.

TABLE 15-1 Traditional versus Agile Time Management

Time Management with Traditional Approaches	Time Management with Agile Approaches
Fixed scope directly drives the schedule.	Scope is not fixed. 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 development, 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 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 development as they learn more about the product and the development team's speed, or <i>velocity</i> . You learn more about velocity later in this chapter.



REMEMBER

A fixed-schedule or fixed-price approach has a 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 scrum teams can deliver products much earlier than traditional project teams. For example, starting development earlier and completing functionality in iterations often allow scrum teams that work with our company, Platinum Edge, to bring value to the market 30 percent to 40 percent faster.



TIP

The reason agile product development efforts finish sooner isn't complicated; they simply start development sooner.

In the next section, find out how to manage time.

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 9.
- » Your detailed planning for each release and at each sprint is tactical. Read more about release planning and sprint planning in Chapter 10.
 - 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 development is underway, use the scrum team's velocity (development speed) to fine-tune your scheduling. We discuss velocity in the next section.



REMEMBER

In Chapter 10, 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 calculate velocity, how to use velocity as a planning input, and ways you might increase velocity throughout product development.

Introducing velocity

One of the most important considerations about time management with agile product development is the use of velocity, a powerful empirical data set that scrum teams use to forecast long-term timelines. *Velocity*, in agile terms, is a development team's work speed. In Chapter 9, we describe estimation of effort for implementing requirements, or user stories, in story points. You measure velocity by the number of user story points that the development team completes, meeting its definition of done, in each sprint.

DETERMINING PRODUCT DEVELOPMENT DURATION

A few factors determine how long agile product development should take:

- **Assigned deadline:** For business reasons, scrum 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 start until the end date.
- **Budget considerations:** Scrum teams may also have budget considerations that affect the amount of time product development will last. For example, if you have a \$1,600,000 budget, and your team costs \$20,000 a week to run, your development effort will last 80 weeks. You'll have 80 weeks to create and release as much shippable functionality as possible. A fixed date can set a limit for your investment decision.
- **Functionality completed:** Agile product development may also last only until enough functionality is complete to achieve some specified amount of value delivered. Scrum 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

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 and implement a user story. Chapter 10 delves into the details of defining 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 development may cost.

In the next section, you dive into velocity as a tool for time management. You see how scope changes affect a timeline. You also find out how to work with multiple scrum teams and review agile artifacts for time management.

Monitoring and adjusting velocity

After development starts, the scrum team starts to monitor its velocity at the end of each 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 actual 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 development 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 estimates and velocity meaningless. Instead, use the scrum team's actual velocity to forecast how much longer the development may take and cost. Also focus on increasing velocity by removing constraints identified during the sprint and at the sprint retrospective. Agile product development is pulled, not pushed.

In the next section, you see how to calculate velocity, how to use velocity to predict a 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. Principle 7 reminds us that “working software (or working product) is the primary measure of progress.” Not velocity.

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.

You don't have to run many sprints before you have real data to forecast. In fact, after you've run only one sprint, you'll have your first empirical data point of the scrum team's velocity. Of course, after you run more sprints, you'll have more empirical data points to use to fine-tune your forecast — based on reality rather than theory.

Using velocity to estimate the development timeline

When you know your velocity, you can determine how long product development 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. 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 additional budget. Even better, the product owner may realize early which features to abandon.

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 400 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, $400/20 = 20$.

If you're using two-week sprints, your product development will last 40 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 product development.
- » 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.



TIP

Another approach to long-range planning is the #noestimates movement, which advocates decomposing product backlog items into equally sized items, rather than estimating items of different story point sizes. *Velocity* in this case refers to how many product backlog items can be accomplished each sprint. Taking the total number of product backlog items in the product backlog and dividing it by the number of product backlog items the team can complete in a sprint (*velocity*) results in a schedule of how many sprints it will take to complete the product.

Velocity differs from sprint to sprint. In the first few sprints, when the product is new, the scrum team will typically have a low velocity. As product development 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

With new teams, velocity will vary considerably from sprint to sprint. Velocity will become more consistent over time, as long as the scrum team members remain consistent. In Chapter 8, we discuss the value of creating long-lived and even permanent scrum teams.

Scrum teams can also increase their velocity, making product development 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 400 story points and an average velocity of 20 story points, product development will last 20 sprints — 40 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 17.39 sprints. If you round that up to 18 sprints, the same product development effort would last 36 weeks.
- » An average velocity of 26 would take about 15.38, rounded up to 16 sprints, or 32 weeks.
- » An average velocity of 31 would take about 12.9, rounded up to 13 sprints, or 26 weeks.

As you can see, even marginally 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. However, opportunities exist to also raise velocity, 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 roadblocks.** One way to increase velocity is to quickly remove 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 impediments in Chapter 11.
- » **Avoid roadblocks.** The best way to increase velocity is to strategically create ways to avoid or prevent 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.



REMEMBER

» **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.

Having a dedicated scrum master who continually helps prevent and 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.



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. As discussed in Chapter 4, they use the scientific method to consistently improve their team.

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.

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 performance only when you use the following practices:

- » **Consistent sprint lengths:** Each sprint should last the same amount of time throughout the life of product development. 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.
- » **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 development, you can use velocity to truly know whether development speed is increasing or decreasing and to accurately estimate the timeline. For this reason, scrum teams adhere to Principle 8: "Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely."



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.



REMEMBER

Shorter is better. If you go from a two-week sprint to a three-week sprint, you have to wait three weeks before you have your first empirical data point. In that same three-week period, going from a two-week sprint to a one-week sprint will give you three new empirical data points.

When you know how to accurately measure and increase velocity, you have a powerful tool for managing time and cost. 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

Scrum teams welcome changing requirements at any time throughout development, which means 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 timeline; you just have to prioritize. Working with the 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 product or may never be created.



REMEMBER

In Chapter 14, we discuss how to manage scope changes with the product backlog. When you add a new requirement, 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 timeline, even with constantly changing scope.

On the other hand, the product owner and the stakeholders may determine that all the requirements in the product backlog, including new requirements, are useful enough to include in the product. In this case, you extend the development end date to accommodate the additional scope, increase velocity, or divide the scope among multiple scrum teams that will work simultaneously on different product features. Learn more about multi-team product development in Chapter 19.

Scrum teams often make schedule decisions about lower-priority requirements toward the end of development. 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. With agile product

development, 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 your work.

The next section shows you how to work with more than one scrum team on a common goal.

Managing time by using multiple teams

For larger development efforts, multiple scrum teams working in parallel may be able to complete development in a shorter time frame.

You may want to use multiple scrum teams if

- » Your development effort is very large and will require more than a single development team of nine or fewer development team members to complete.
- » Your development effort 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 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.

In Chapter 19, 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 15–2 shows how each artifact contributes to time management.

In the next sections, you dive into cost management for agile product development. Cost management is directly related to time management. You compare traditional approaches to cost management to those with agile product development. You find out how to estimate costs and how to use velocity to forecast your long-term budget.

TABLE 15-2

Agile Artifacts and Time Management

Artifact	Role in Time Management
Product roadmap: 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 Chapter 9.	The product roadmap is a strategic look at the overall product 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.
Product backlog: The product backlog is a complete list of all currently known product requirements. Find more about the product backlog in Chapters 9 and 10.	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 end date.
Release plan: The release plan contains a release schedule for a minimum set of requirements. Find more about the release plan in Chapter 10.	The release plan will have a target release date for a specific goal supported by a minimal set of marketable features. Scrum teams plan and work on only one release at a time.
Sprint backlog: The sprint backlog contains the requirements and tasks for the current sprint. Find more about the sprint backlog in Chapter 10.	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.

What's Different about Agile Cost Management?

Cost is a product's financial budget. When you work with an agile product development approach, 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 product development takes a different approach to budget and cost management than traditional projects. Table 15-3 highlights some of the differences.



TECHNICAL STUFF

When costs increase, 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.

TABLE 15-3 Traditional versus Agile Cost Management

Cost Management with Traditional Approaches	Cost Management with Agile Approaches
Cost, like time, is based on fixed scope.	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 funding after the product roadmap stage is complete. Some organizations even fund 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.	Scrum teams can replace lower-priority requirements with new, equivalently sized high-priority requirements with no effect on time or cost.
Scope bloat (see Chapter 12) 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 development.
Projects cannot generate revenue until the project is complete.	Scrum teams can release working, revenue-generating functionality early, creating a self-funding product.

In the following sections, you find out about cost approaches to agile product development, how to estimate costs, how to control your budget, and how to lower costs.

Managing Agile Budgets

With agile product development, cost is mostly a direct expression of 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 remaining sprints — that is, how long your product development effort will be — you can know how much your product development effort will cost.

Cost also includes the cost for resources like hardware, software, licenses, and any other supplies you might need to complete development.

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 the budget for your product, you need to know the cost for your scrum team, per sprint, and the cost for any additional resources you need to complete development.

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 15-4 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 15-4 Sample Scrum Team Budget for a Two-Week Sprint

Team Member	Hourly Rate	Weekly Hours	Weekly Cost	Sprint Cost (2 Weeks)
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. In addition to scrum team member costs, take the following into account when determining your 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 development. (To keep calculations simple throughout this chapter, we assume that the per-sprint cost is \$40,000, which 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, 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 product

A big benefit of agile product development is the capability to become self-funding. 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 product development.

For example, an ecommerce website that holds all completed work for one big release after six months instead of releasing functionality incrementally may be able to generate \$100,000 per month. However, the same ecommerce website might be able to start generating \$15,000 a month in sales after the first release of some essential valuable functionality, \$40,000 a month after the second incremental release of valuable functionality, and so on. Tables 15–5 and 15–6 compare income on a sample traditional project to the income from a self-funding agile product development effort.

In Table 15–5, the project created \$100,000 in income after six months of development. Now compare the income in Table 15–5 to the income generated in Table 15–6.

In Table 15–6, the product generated income with the first release. By the end of six months, the product had generated \$330,000 — \$230,000 more than the project in Table 15–5.

TABLE 15-5

Income from a Traditional Project with a Final Release after Six Months

Month	Income Generated	Total Project Income
January	\$0	\$0
February	\$0	\$0
March	\$0	\$0
April	\$0	\$0
May	\$0	\$0
June	\$0	\$0
July	\$100,000	\$100,000

TABLE 15-6

Income with Monthly Releases and a Final Release after Six Months

Month/Release	Income Generated	Total Income
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

Using velocity to determine long-range costs

The “Using velocity to estimate the development timeline” section, earlier in this chapter, shows you how to determine how much time product development 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 of developing the entire product or just the current release.

After you know the scrum team’s velocity, you can calculate the cost for the remainder of product development.

In the velocity example from earlier in this chapter, where your scrum team velocity averages 16 story points per sprint, your product backlog contains 400 story points, and your sprints are 2 weeks long, your product will take 25 sprints, or 50 weeks, to complete.

To determine the remaining cost of product development, 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 25 sprints left, your remaining cost for development will be \$1,000,000.

In the next sections, you find out different ways to lower your 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 15-4, 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 20 remaining sprints.
 - Your remaining development will cost \$800,000, saving you \$200,000.
- » If the scrum team increases its velocity to 23 story points
 - You will have 18 remaining sprints.
 - Your remaining development will cost \$720,000, saving you an additional \$80,000.
- » If the scrum team increases its velocity to 26 story points
 - You will have 16 remaining sprints.
 - Your remaining development will cost \$640,000, an additional \$80,000 savings.

As you can see, increasing the scrum team's velocity by removing impediments can provide real savings. 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 costs by not completing lower-priority requirements, thus lowering the number of sprints you need. Because completed functionality is delivered with each sprint with agile product development, the stakeholders can make a business decision to end development when the cost of future development is higher than the value of that future development.

Stakeholders can then use the remaining budget from the canceled development effort to develop something even more valuable. The practice of moving the budget from one development effort to another is called *capital redeployment*.

To determine when development should 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 something new

When $V < AC + OC$, product development can stop because the cost you'll sink into the remaining product requirements will be more than the value you will receive from them.

For example, consider the following about a company using agile product development techniques:

- » 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 product initiative that would generate \$150,000 after three sprints, minus the scrum team's cost ($OC = \$150,000$).
- » The product value, \$100,000, is less than the actual costs plus opportunity costs, or \$270,000. This would be a good time to stop and redeploy capital to the next product initiative.

The opportunity for capital redeployment sometimes arises in emergencies, when an organization needs the scrum team to pause or pivot to something more valuable or both. Sponsors sometimes evaluate the remaining value and cost before restarting new product development.



WARNING

Pausing and pivoting can be expensive. The costs associated with demobilization and remobilization — saving work in progress, documenting current state, debriefing paused team members, retooling for the new development, briefing team members on the new development, learning new skills required on the new development — can be significant and should be evaluated before making the decision to pause development that may need to be remobilized again in the future. $V < AC + OC$ can help with this decision.

Sponsors may also compare the product backlog value to remaining development costs throughout product development, so they know just the right time to end development 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 development. 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. Refer to Table 15–2 to see how each artifact helps you measure and evaluate development time and costs.

Time and cost forecasts based on the team's empirically proven development pace are more accurate than forecasts based on hypotheticals or what the team hopes to accomplish.

IN THIS CHAPTER

- » Recognizing how agile principles change team dynamics
- » Understanding how communication differs with agile product development
- » Seeing how communication works

Chapter **16**

Managing Team Dynamics and Communication

Team dynamics and communication are significant parts of agile product development. In this chapter, you find out about traditional and agile approaches to teams and communication. You see how a high value on individuals and interactions makes agile teams great teams to work on. You also find out how face-to-face communication helps make agile product development successful.

What's Different about Agile Team Dynamics?

What makes an agile team 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 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 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 product management areas. You see some of these principles repeated in different chapters of this book.



REMEMBER

With agile product development, 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 product team is the scrum team and your stakeholders. Everyone on the scrum team has responsibilities related to self-management.

Table 16-1 shows some differences between team management with traditional projects and agile product development.



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 team are human beings, with emotions, ideas, and priorities inside and outside. People can learn and create and grow throughout their experience of working together. Respecting your fellow 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 product development. 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 product development succeed.

TABLE 16-1 Traditional versus Agile Team Dynamics

Team Management with Traditional Approaches	Team Dynamics with Agile Approaches
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 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 goal at a time, and reap the benefits of focus.
Development team members have distinct roles, such as programmer or tester.	Agile organizations focus on skills instead of titles. 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> . With agile product development, you probably will not hear the term <i>resource</i> used to refer to people.

Managing Team Dynamics

Time and again, when we talk with product owners, developers, and scrum masters, we hear the same thing: People enjoy agile product development. Scrum 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 development successful.

Becoming self-managing and self-organizing

With agile product development, 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 development efforts have no leadership. Each member of the scrum team has the opportunity to lead informally, based on his or her skills, ideas, and initiative.

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 product development. The difference is that agile 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 product. 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 product 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 development. Table 16-2 shows how scrum teams and development teams manage scope, procurement, time, cost, team dynamics, communication, stakeholders, quality, and risk.

TABLE 16-2**Product Management and Self-Managing Teams**

Area of Product Management	How Product Owners Self-Manage	How Development Teams Self-Manage	How Scrum Masters Self-Manage
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	<p>Ensure that the development team correctly understands product features so that development teams can correctly estimate the effort to create those features.</p> <p>Use velocity — development speed — to forecast long-term timelines.</p>	<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 each sprint.</p> <p>Choose their own daily schedules and manage their own time.</p>	<p>Facilitate estimation poker games.</p> <p>Help development teams increase velocity, which affects time.</p> <p>Shield the team from organizational time-wasters and distractions.</p>
Cost	<p>Ultimately responsible for the budget and return on investment.</p> <p>Use velocity to forecast long-term costs, based on timelines.</p>	Provide effort estimates for product features.	<p>Facilitate estimation poker games.</p> <p>Help development teams increase velocity, which affects cost.</p>

(continued)

TABLE 16-2 (continued)

Area of Product Management	How Product Owners Self-Manage	How Development Teams Self-Manage	How Scrum Masters Self-Manage
Team dynamics	<p>Commit to their product 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 product 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>Servant-leaders who 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 development progress to stakeholders.</p> <p>Help present working functionality to stakeholders at the sprint review meetings at the end of each sprint.</p>	<p>Inspect progress, coordinate 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 development status.</p> <p>Present working functionality to 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 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>

Area of Product Management	How Product Owners Self-Manage	How Development Teams Self-Manage	How Scrum Masters Self-Manage
Quality	<p>Add and clarify acceptance criteria to requirements.</p> <p>Ensure that the development team correctly understands and interprets requirements.</p> <p>Provide development teams with feedback about the product from the organization and from the marketplace.</p> <p>Accept functionality as done during each sprint.</p>	<p>Commit to providing technical excellence and good design.</p> <p>Test their work throughout the day and comprehensively test all development each day.</p> <p>Inspect their work and adapt for improvements at sprint retrospective meetings at the end of each sprint.</p>	<p>Help facilitate the sprint retrospective.</p> <p>Help ensure face-to-face communication between scrum team members, which in turn helps ensure quality work.</p> <p>Help create a sustainable development environment so that the development team can perform at its best.</p>
Risk	<p>Look at overall product risks as well as risks to their ROI commitment.</p> <p>Prioritize high-risk items on the product backlog near the top to address them sooner rather than later.</p>	<p>Identify and develop the risk mitigation approach for each sprint.</p> <p>Alert the scrum master to roadblocks and distractions.</p> <p>Use information from each sprint retrospective to reduce risk in future sprints.</p> <p>Embrace cross-functionality to reduce risk if one member unexpectedly leaves the team.</p> <p>Commit to delivering shippable functionality at the end of each sprint, reducing overall product risk.</p>	<p>Help prevent roadblocks and distractions.</p> <p>Help remove roadblocks and identified risks.</p> <p>Facilitate development team conversations about possible risks.</p>



All in all, people developing products using agile techniques 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 who use agile product development techniques 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. Agile leaders 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 during product development. The servant leadership mindset permeates the entire team.

Larry Spears identified ten characteristics of a servant-leader in his paper, “The Understanding and Practice of Servant-Leadership” (Servant Leadership Round-table, 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.

- » **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:** 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:** 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. The changing nature of agile lifecycles 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 product 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 product development. Much like self-management, servant leadership requires trust and respect.



TECHNICAL
STUFF

The concept of servant leadership is not specific to agile product development. 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 scrum team success: the dedicated team and the cross-functional team.

Working with a dedicated team

Having a dedicated scrum team provides the following important benefits:

- » **Keeping people focused on one goal at a time helps prevent distractions.** Dedication to one goal, such as a sprint goal, 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 initiative, that person has the time and clarity to ensure his or her work is the best it can be. Chapter 17 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.** 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 goal, 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 15, 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 development effort 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 objectives or goals at

one time, under the mistaken assumption that they 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.



WARNING

Work in progress is expensive inventory and creates no value. Scrum teams continually seek to reduce work in progress through focus and dedication.

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 product return on investment, so be willing to fight for your product's success.
- » If you're a member of the development team and anyone requests that you do work outside the current sprint goal, 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 potentially expensive distraction.
- » If you're a scrum master, as the expert on agile approaches, you can educate the company on why a dedicated scrum team means decreased work in progress and 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 development efforts. See Chapter 8 to learn more about the importance of stable, long-lived, and even permanent teams.

Another characteristic of scrum teams is that they are cross-functional.

Working with a cross-functional team

Cross-functional development teams are also important. The development team doesn't just include programmers — people writing software code; it includes all the people who will have a job during development to transform a product requirement into something that is valuable and shippable. For non-software scrum teams, there won't be any programmers, but there will be people with the range of skills needed to create the product.

For example, a scrum team developing software would include people with programming, database, quality assurance, usability, graphics, design and infrastructure skills. While each person has specialties, being cross-functional means that everyone on the team is willing to pitch in on different parts of development, as much as possible. The same is true for non-software products.

As a member of a 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 a development team, but you should be willing to learn new skills and help with all kinds of tasks. To learn more about building individual and team capability, see the T-shaped, pi-shaped and M-shaped models discussed in Chapter 7.



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 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 team 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 with agile development. Working on a 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 development.

Both the physical environment and the cultural environment of an organization are important keys to success. 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 16-1 illustrates the difference between an email exchange and a conversation in person.

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 6 provides details on how to set up both physical and virtual environments for a scrum team to communicate effectively.

Having a cultural environment of openness, which is conducive to scrum team growth, is another success factor. 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.
- » 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.

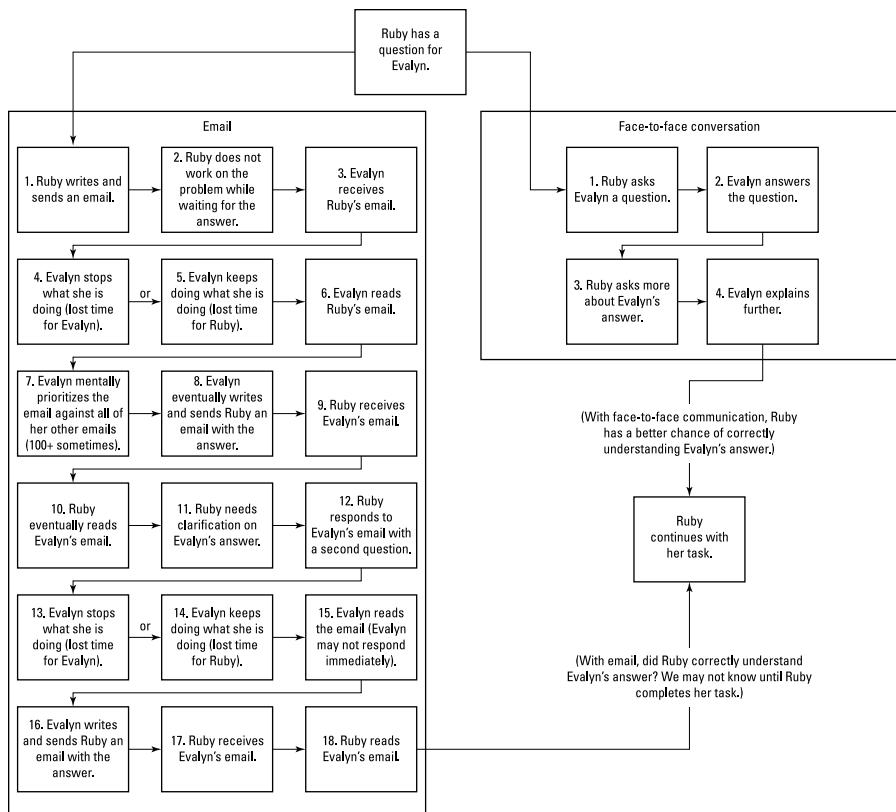


FIGURE 16-1:
Email versus
face-to-face
conversation.

Trust, openness, and respect are fundamental to team dynamics.



TIP

Some of the best product and process improvements come from novices asking “silly” questions.

Another facet of scrum team dynamics is the concept of the size-limited team.

Limiting development team size

An interesting psychological aspect of team dynamics is the number of people on a development team. Scrum 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.

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 team.



TIP

If your product development requires more than nine development team members or you think you need to create a position to improve intra-team communication, consider breaking up the work between multiple scrum teams instead. Find details on how to work with multiple scrum teams in Chapters 15 and 19.

Managing product development with dislocated teams

As we say throughout the book, a collocated scrum team is ideal for agile product development. 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 physical locations), as we all witnessed during the COVID-19 pandemic, exist for many reasons and in different forms.

In some companies, people with the right skills for a team may work in different offices, and the company may not want the cost of bringing those people together for the duration of development. Some organizations work jointly with other organizations on development, 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 products with people from other countries.



TIP

If you need to offshore, then offshore with both feet. Collocate entire scrum teams with the product owner, development team, and scrum master together. You may have one scrum team in one physical location, and another at a different physical location, but organize your scrum teams to be together.

The good news is that you can still develop 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. Dislocated teams are often more effective taking an agile approach by using scrum than without it.

Table 16–3, from Amblysoft’s “Agile Adoption Rate Survey Results” in 2008, shows a comparison of success rates for products developed with collocated scrum teams against those with geographically dispersed scrum teams. Even for teams separated by large distances, with scrum they still have high success rates.

TABLE 16-3

Success of Collocated and Dislocated Scrum Teams

Team Location	Success Percentage
Collocated scrum team (everyone on the team is in the same physical location)	83%
Dislocated but physically reachable (team members work in different physical locations but can travel to be face to face)	72%
Distributed across geographies (team members are separated significantly, for example, by time zones)	60%

“Agile Adoption Rate Survey Results” (Scott W. Ambler, Amblysoft, Copyright © 2008)

How do you successfully develop a product with a dislocated scrum team? We have three words: communicate, communicate, and communicate. Because daily in-person conversations are not possible throughout the entire day, dislocated scrum teams require unique efforts by everyone working on the product. 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 liberally throughout the day, not just for sprint meetings. Make sure team members are ready for impromptu video chats and have the necessary equipment, such as sufficient bandwidth, microphones, headphones and multiple monitors, to make videoconferencing successful.
- » **If possible, arrange for the scrum team members to meet in-person in a central location at least once at the beginning of the development, and preferably multiple times throughout.** 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 work moving. For drastic time zone differences, consider trading off on times you are available. One week, Location A can be available in the early morning. The next, Location 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. Additional communication effort is required for a successful dislocated team.
- » **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, which is another reason to meet in-person in each other's physical locations.
- » **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 development can succeed.

The unique approaches to team dynamics is part of what make agile product development successful. Communication is closely related to team dynamics, and agile communication methods 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 product development.

However, the agile principles set a different tone, 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 is more important.

Table 16-4 shows some differences between communication on traditional projects and agile development.



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.

TABLE 16-4

Traditional versus Agile Communication

Communication Management with Traditional Approaches	Communication Management with Agile Approaches
Team members might make no special effort for in-person conversations.	Agile 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 status at a glance. Teams use the <i>show, don't tell</i> concept, showing working functionality 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 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.

Managing Agile Communication

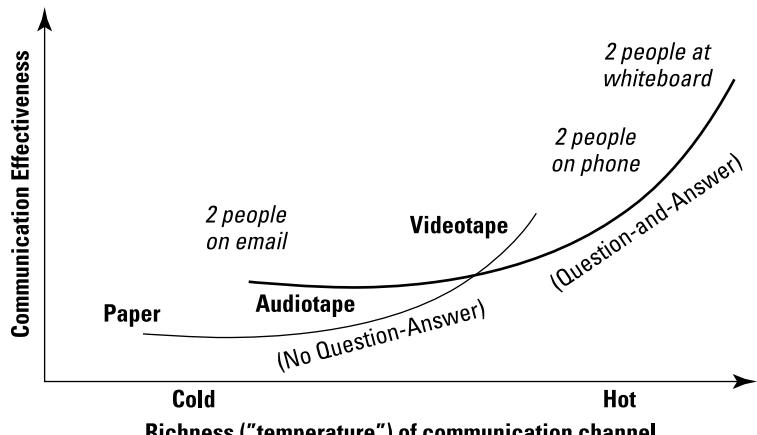
To manage communication with agile product development, you need to understand how different agile communication methods work and how to use them together. You also need to know why status is different and how to report progress to stakeholders. The following sections show you how.

Understanding agile communication methods

You can communicate through artifacts, meetings, and informally.

Face-to-face conversations are the heart and soul of agile product development. When scrum team members talk with one another about the product 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 16-2, from Alistair Cockburn's presentation *Software Development as a Cooperative Game*, shows the effectiveness of face-to-face communication versus other types of communication.



In previous chapters, we describe a number of artifacts and meetings that fit with agile development. All the agile artifacts and meetings play a role in communication. Agile meetings provide a format for communicating in a face-to-face environment. Agile meetings 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 16-5 provides a view of the different communication channels with agile product development.

TABLE 16-5 Agile Communication Channels

Channel	Type	Role in Communication
Release planning and sprint planning	Meetings	Planning meetings have specific desired outcomes or are focused on a specific scope of work driven by a business goal, and concisely communicate the purpose and details of the release, and the sprint to the scrum team. Learn more about planning meetings in Chapters 9 and 10.
Product vision statement	Artifact	The product vision statement communicates the end goal of the product to the team and the organization. Find out more about the product vision in Chapter 9.
Product roadmap	Artifact	The product roadmap communicates a long-term view of the features that support the product vision and are likely to be part of the product. Find out more about the product roadmap in Chapter 9.

Channel	Type	Role in Communication
Product backlog	Artifact	The product backlog communicates the scope of the product as a whole to the team. Find out more about the product backlog in Chapters 9 and 10.
Release plan	Artifact	The release plan communicates the goal and timing for a specific release. Find out more about the release plan in Chapter 10.
Sprint backlog	Artifact	When updated daily, the sprint backlog provides immediate sprint and development 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 Chapters 10 and 11.
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. As a team member moves a completed task, everyone knows it's time for the next task. Find out more about the task board in Chapter 11.
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 Chapter 11.
Face-to-face conversations	Informal	Face-to-face conversations are the most effective mode of communication.
Sprint review	Meeting	The sprint review is the embodiment of show, don't tell, philosophy. Displaying working functionality to the entire team conveys progress in a more meaningful way than a written report or a conceptual presentation ever could. Find out more about sprint reviews in Chapter 12.
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 Chapter 12.
Meeting notes	Informal	Meeting notes are an optional, informal communication method. 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 are updated product backlog items. Notes from a sprint retrospective are action items added to the product backlog for consideration in a future sprint and 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. Digital versions of these can be effective.



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. Agility is about people and interactions; tools are secondary to success.

The next section addresses a specific area of agile communication: status reporting.

Status and progress reporting

All products have stakeholders, people outside the immediate scrum team who have a vested interest in the product. At least one of the stakeholders is the person responsible for paying for your development (the sponsor). It is important for stakeholders, especially those responsible for budgets, to know how development is progressing. This section shows how to communicate your status.

Status for scrum teams is a measure of the features that the team has completed. Using the definition of done from Chapters 2, 10, 12, and 17, 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 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.

Working functionality that meets the definition of done is the primary measure of progress. You can confidently say that product features are complete. Because scope changes constantly, 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 release 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 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. Principle 7 says, "Working software is the primary measure of progress." Show, don't tell.

Strongly encourage anyone who may have an interest in your product 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

Don't get sucked in to *double work agile*. 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 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 adoptions. Scrum teams will burn out quickly if they try to meet the demands of two drastically different development 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.

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. The sprint backlog provides the daily status, so you don't need to spend time on status in your daily scrum. Your daily scrum is for coordinating work for the day, not for status.



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 product development. 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 11, we show you an image of sample burndown charts for different sprint scenarios; here it is again in Figure 16–3.

If you update your sprint backlog every day, you'll always have an up-to-date status for your 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.

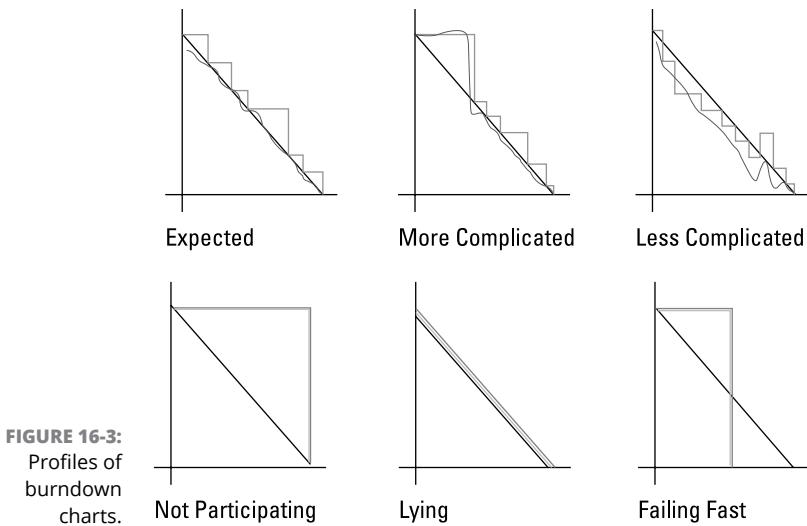


FIGURE 16-3:
Profiles of
burndown
charts.



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 team the status of a sprint, release, or even of the entire product. 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 development stands, because the team sees the task board every day. See an example of a task board in Chapter 6.

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.

IN THIS CHAPTER

- » Learning how agile quality approaches reduce risk
- » Discovering ways to ensure quality product development
- » Taking advantage of automated testing for better productivity
- » Understanding how agile development approaches reduce risk

Chapter 17

Managing Quality and Risk

Quality and risk are closely related parts of product development. In this chapter, you find out how to deliver quality products using agile methods. You understand how to take advantage of agile approaches to manage product risk. You see how quality has historically affected risk, and how agile quality management fundamentally reduces risk.

What's Different about Agile Quality?

Quality refers to whether a product works, and whether it fulfills the stakeholders' and customers' needs. Quality is an inherent part of agile product 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 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 stakeholders.

Table 17-1 shows some differences between quality management on traditional projects and agile product development.

BUGS. BUGS? BUGS!

Why do we call computer problems *bugs*? The first computers were large, glass-encased machines that took up entire rooms. In 1947, 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 was on display for years at the Smithsonian National Museum of American History.

TABLE 17-1 Traditional versus Agile Quality

Quality Management with Traditional Approaches	Quality Dynamics with Agile Approaches
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 in software development include face-to-face communication, pair programming, test-driven development (also known as test-first development), 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 <i>bugs</i> 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 frequently and incrementally test smaller amounts of shippable 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 because it is part of every sprint to satisfy the team's definition of done.

At the start of this chapter, we state that quality and risk are closely related. The agile approaches in Table 17-1 greatly reduce the risk and unnecessary cost that usually accompany deferred quality management.

Another difference about quality is the multiple quality feedback loops throughout development. In Figure 17-1, you see the different types of product feedback a scrum team receives in the course of development. The development team can immediately incorporate this feedback into the product, increasing product quality on a regular basis.

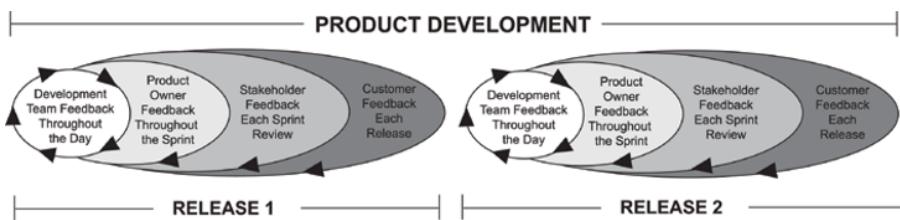


FIGURE 17-1:
Quality feedback cycles.



REMEMBER

In Chapter 16, we tell you that development teams include everyone who works on a product. Development teams must 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 development. Cross-functionality extends to quality activities such as preventing issues, testing, and fixing bugs.

In the next section, you see how to use agile techniques to increase quality.

Managing Agile Quality

Agile development teams have the primary responsibility for quality. 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-prefixes 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.” QA usually refers to whether the product meets the business or customer need for which it was created. *Quality control* (QC) is also a common way to refer to quality management, but it refers more directly to the technical quality of the product.

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 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 development 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 agile development.

Quality and the sprint

Quality management is a daily part of agile product development. Scrum teams run development 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 10, 11, and 12.



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 a single backlog item's worth of development than in an entire product. Iterative development makes quality product development easier.

Scrum teams test throughout each sprint. Figure 17-2 shows how testing fits into sprints. Notice that testing begins in the first sprint, as soon as the development team starts creating the first requirement.

When development teams test throughout each sprint, they can find and fix defects very quickly. Development teams implement 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 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 product development 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

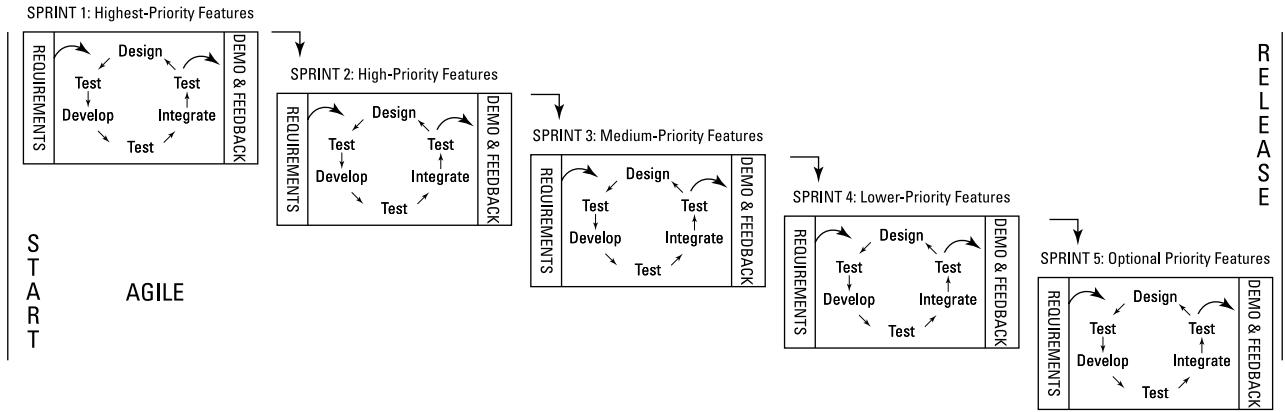


FIGURE 17-2: Testing within sprints.

The following sections provide a detailed look at each of these proactive quality practices.



REMEMBER

Continuous attention to technical excellence and good design

Scrum 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 product 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 for a company that believes in and rewards technical excellence, enacting this agile principle will be easy.

Other companies may undervalue technical excellence; scrum 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 5.



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 development effort, 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 execute one development task. 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 work as soon as it is completed. 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 the product.

- » **Collective code ownership:** In this approach, everyone on the development team can create, change, or fix any part of the code. 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 it 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; you need to create a code build with each check-in before running automated tests. Find out more about automated testing later in this chapter.



REMEMBER

With agile product development, the development team decides which tools and techniques will work best for the sprints, the product, and the 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 approaches. We encourage you to learn more about these, 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 24 for more recommendations.

The product owner and development team

Another aspect of agile product development 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 wasted time and effort.



REMEMBER

The product vision statement not only articulates the product's goals but also communicates how your product supports the company's or organization's strategies. Chapter 9 explains how to create a product vision statement.

User stories and acceptance criteria

Another proactive quality measure with agile development is the acceptance criteria you build into each user story. In Chapter 9, 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 17-3 shows a user story and its acceptance criteria.

FIGURE 17-3:
A user story and
acceptance
criteria.

Title Transfer money between accounts	When I do this: When I view my account balances,	This happens: I see an option to transfer funds.
As Carol,	When I select the transfer option,	I choose between which accounts I want to transfer funds.
I want to transfer funds between accounts	When I select the "transfer from" option,	I see a list of my available accounts and balances.
so that 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.
<hr/> Value	<hr/> Jennifer	<hr/> Estimate

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 16, 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 because it leads to better interpretation of requirements, roadblocks, and discussions between scrum team members.

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.

With agile development, scrum teams help ensure that they do quality work by creating an environment where members of the development team sustain a constant working pace indefinitely. 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 agile development, you look at both your product and your process (inspect)

and make changes as necessary (adapt). Chapters 9 and 12 have more information about this tenet.

In the sprint review and sprint retrospective meetings, scrum teams regularly step back and review their work and methods and determine how to make adjustments for a better product. We provide details on the sprint review and sprint retrospective in Chapter 12. Following is a quick overview of how these meetings help ensure quality.

In a sprint review, scrum teams review requirements completed at the end of each sprint. Sprint reviews address quality by letting stakeholders see working requirements and provide feedback on those requirements throughout the course of development. 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, the development process, or work environment that might increase quality.

The sprint review and sprint retrospective aren't the only opportunities for inspecting and adapting for quality. Agile approaches encourage reviewing work and adjusting behavior and methods throughout each workday. Daily inspecting and adapting everything you do on the product help ensure quality.

Another way to manage and help assure quality is to use automated testing tools. The next section explains why automated testing is important to agile software development and how to incorporate automated testing into your product.

Automated testing

Automated testing is the use of software to test your product. If you want to quickly create functionality that meets the definition of done — designed, developed, 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. Scrum 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

Teams won't become agile without automated testing. Manual testing simply takes too long.

Throughout this book, we explain how scrum 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 defects than manual testing.

To develop a product using automated testing, development teams develop and test using the following steps:

- 1. While implementing user stories, write and execute automated tests in support of user stories.**
- 2. Whenever something is added, run automated tests as part of the organization's continuous integration and continuous deployment pipeline.**
- 3. Give feedback to developers immediately when tests fail, ensuring that the errors are corrected immediately.**



TIP

Automated testing allows development teams 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 product test. If you're new to product development, you may not know that there are many different types of 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 a product performs given different scenarios.
- » **Load testing:** Tests how well a product handles different amounts of concurrent activity.
- » **Security testing:** Tests for product vulnerabilities, nefarious threats, and weaknesses that can be exploited.
- » **Smoke testing:** Tests on small but critical parts to help determine if the product as a whole is likely to work.
- » **Static testing:** Focuses on checking standards, rather than working product.

Automated testing works for these tests as well as many other types of product tests.

As you may understand by now, quality is an integral part of agile product development. Quality is just one factor, however, that differentiates risk with agile product development from traditional projects. In the next sections, you see how risk on traditional projects compares to risk with agile development.

What's Different about Agile Risk Management?

Risk refers to the factors that contribute to a project's success or failure. With agile product development, 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.
7. 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 development efforts are 30 percent more likely to succeed than traditional projects. See Figure 17-4. 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. In other words, the bigger the development effort, the higher the chance of success using agile approaches.

Table 17-2 shows some of the differences between risk on traditional projects and agile development.

With agile approaches, risk declines as development progresses. Figure 17-5 shows a comparison of risk and time between waterfall projects and agile product development.

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%

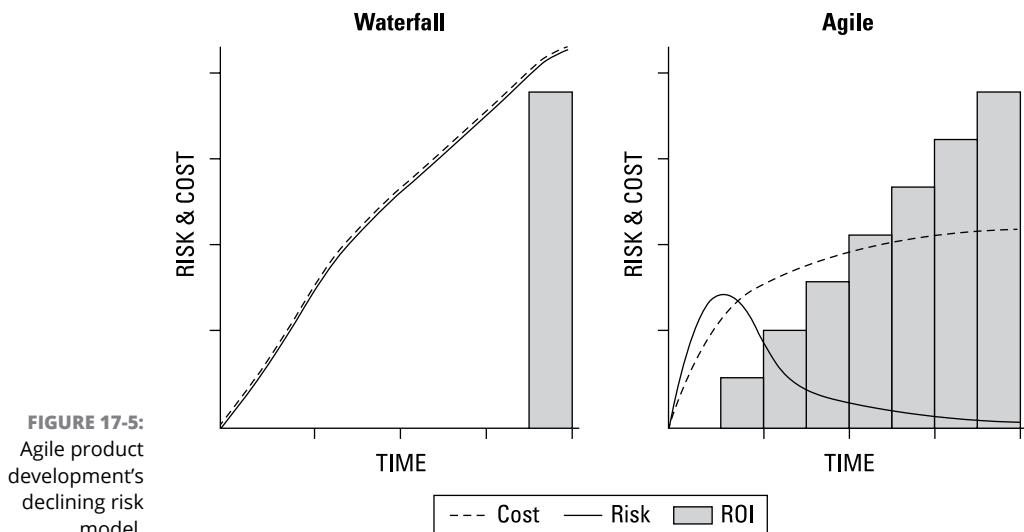
FIGURE 17-4:
Standish Group's
"2015 Chaos Report."

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.

TABLE 17-2 Traditional versus Agile Risk

Risk Management with Traditional Approaches	Risk Dynamics with Agile Approaches
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 and incrementally every sprint, rather than sinking costs into a product 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 failed development.
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 techniques accommodate new high-priority requirements without increasing time or cost by removing a low-priority requirement of equal time and cost.

Risk Management with Traditional Approaches	Risk Dynamics with Agile Approaches
Traditional projects require precise 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.	Time and cost is estimated using the scrum team's actual performance, or velocity. You refine estimates throughout development, because the longer you work, the more you learn about the product, 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 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.



All projects have some risk, regardless of your approach. However, with agile product 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 agile product development. In the next section, you see why.

Managing Agile Risk

In this section, you examine key structures of agile development that reduce risk over the life of the product. You find out how to use agile tools and events to find risks at the right time 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 product investment and proof that the product works. Sprints also provide the potential for a product 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 with agile development are the definition of done, self-funding, 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 12, 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 categories:

- » **Developed:** The development team must fully create the working product requirement in an environment that reflects the one where the customer will use it.
- » **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 17–6 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 17-6:
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 with agile approaches. By creating a product that meets the definition of done in every sprint, you end each sprint with a working, usable, and shippable product increment. Even if outside factors cause development to end early, stakeholders will always see some value and have working functionality to use now and build upon later.

Self-funding development

Agile development efforts can mitigate financial risk in a unique way that traditional projects cannot: self-funding development. Chapter 15 includes examples of self-funding development. If your product is an income-generating product, you could use that income to help fund the rest of your product.

In Chapter 15, we show you two different ROI models. Here they are again, in Tables 17-3 and 17-4. The development efforts in both tables create identical products.

In Table 17-3, the product created \$100,000 in income after six months of development. Now compare the ROI in Table 17-3 to the ROI in Table 17-4.

TABLE 17-3

Income from a Traditional Project with a Final Release after Six Months

Month	Income Generated	Total Project Income
January	\$0	\$0
February	\$0	\$0
March	\$0	\$0
April	\$0	\$0
May	\$0	\$0
June	\$0	\$0
July	\$100,000	\$100,000

TABLE 17-4

Income from Agile Development with Monthly Releases and a Final Release after Six Months

Month/Release	Income Generated	Total Income
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 17-4, the product generated income with the very first release. By the end of six months, the product had generated \$330,000 — \$230,000 more than the project in Table 17-3.

The capability to generate income in a short amount of time has a number of benefits for companies and teams. Self-funding agile development 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 up front. For groups short on cash, self-funding can enable product features that would otherwise not be feasible.

Self-funding also helps mitigate the risk that development 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—with nothing tangible to show for the time spent to that point. However, agile product development that generates additional revenue with every release has a good chance of continuing during a crisis.

Finally, self-funding helps sell stakeholders in the first place; it's hard to argue with a product that provides continuous value and pays for at least part of the product 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 product from moving forward during the testing phase, development teams identify critical problems within a few sprints. This quantitative risk mitigation can save organizations large amounts of money.

Tables 17–5 and 17–6 illustrate the difference in sunk costs for a failed waterfall project and a failed agile development effort. Both tables are for identical products with identical costs.

In Table 17–5, the 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 17–5 to that in Table 17–6.

By testing early (daily during each two-week sprint), the development team from Table 17–6 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 17–5.



REMEMBER

Because of the definition of done, product development that fails early still leaves you with something tangible that you may leverage or improve. For example, the failed development effort in Table 17–5 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 end the product development 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 development, 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.

TABLE 17-5 Cost of Failure on a Waterfall Project

Month	Phase and Issues	Sunk Project Cost	Total Sunk Project Cost
January	Requirements phase	\$80,000	\$80,000
February	Requirements phase	\$80,000	\$160,000
March	Requirements phase	\$80,000	\$240,000
April	Requirements phase	\$80,000	\$320,000
May	Design phase	\$80,000	\$400,000
June	Design 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 17-6 Cost of Failure with Agile Techniques

Month	Sprint and Issues	Sunk Cost	Total Sunk Cost
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	Development cancelled; product not viable.	0	\$160,000

Finally, failing fast does not necessarily mean 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 development, and the idea of failing fast, along with the foundation of agile principles, all help lower risk. In the next section, you see how to actively use agile tools to manage risk.

Identifying, prioritizing, and responding to risks early

Although the structure of agile product development inherently reduces many traditional risks, development teams still should be aware of the problems that can arise during development. 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

Scrum teams prioritize the highest-value and highest-risk requirements first.

Instead of spending hours or days documenting all of the 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 product and problems that are more likely to arise. Table 17-7 shows how scrum teams can use the different agile tools to manage risk at the right time.

TABLE 17-7 Agile Risk Management Tools

Artifact or Meeting	Role in Risk Management
Product vision	<p>The product vision statement helps unify the team's definition of product goals, mitigating the risk of misunderstandings about what the product will need to accomplish.</p> <p>While creating the product vision, the 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 Chapter 9.</p>
Product roadmap	<p>The product roadmap provides a visual overview of the product's requirements and priorities. This overview allows the team to quickly identify gaps in requirements and incorrectly prioritized requirements. Find out more about the product roadmap in Chapter 9.</p>
Product backlog	<p>The product backlog is a tool for accommodating change in the product. 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.</p> <p>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 Chapters 9 and 10.</p>

(continued)

TABLE 17-7 (continued)

Artifact or Meeting	Role in Risk Management
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 Chapter 10.
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 Chapter 10.
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 status in Chapter 11.
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 Chapter 11.
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 11.
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 development with the wrong product. Find out more about sprint reviews in Chapter 12.
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 Chapter 12.

The artifacts and meetings discussed in this section systematically help scrum teams manage risk with agile development by addressing risk by the responsible roles at appropriate times. The larger and more complex the product, the higher the likelihood that an agile approach can eliminate the risk of failure.

Ensuring Success

IN THIS PART . . .

Build a foundation through organizational and individual commitment to becoming more agile.

Choose your first agile product development opportunity and create an environment that will optimize agile transition success.

Simplify agile techniques across multi-team product development, enabling alignment and autonomy.

Become a change agent in your organization and help avoid both organizational and leadership pitfalls in agile transitions.

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 organizational support

Chapter **18**

Building a Foundation

To successfully move from traditional project management 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 team for your first pilot of agile techniques, providing the team an environment conducive to agile approaches. You want to find the right training for your team, and sustainably support your organization's agile approach so that it can grow beyond your first development effort.

In this chapter, we show you how to build a strong agile foundation within your organization.

Organizational and Individual Commitment

Commitment to agile product development 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 scrum team members may find themselves forced back into old project management processes. Without the commitment of individual 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 team members.

Organizations can commit to an agile transition by doing the following:

- » Training leaders in skills specific to leading the transformation internally
- » Engaging an experienced agile expert to create a realistic transition plan based on the current state and to guide the company through that plan
- » Investing in employee training, starting with the members of the company's first scrum 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 are dedicated: An empowered product owner, a cross-functional development team of multi-skilled people, and an influential servant-leader scrum master
- » Enabling development teams to continuously increase their skill sets
- » Providing automated testing tools and a continuous integration framework
- » Logistically supporting scrum team collocation for effective and real-time collaboration
- » Ensuring that distributed teams are organized in similar time zones and making an investment in appropriate virtual tools and training when rare circumstances prevent physical collocation, such as the COVID-19 pandemic
- » Allowing scrum teams to manage themselves
- » Empowering product owners to make business prioritization decisions, development teams to own technical excellence decisions, and scrum masters to challenge the status quo by breaking down organizational constraints to agility

- » Encouraging learning by giving scrum teams the time and freedom to go through a healthy trial-and-error process
- » Revising employee performance reviews to emphasize team performance
- » Encouraging scrum 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 scrum 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 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 scrum team works at adopting agile practices, the changes become easier for everyone on the 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
- » Letting go of ego and working as part of a team, especially across traditional hierachal and departmental boundaries
- » Resisting the temptation to fall back on old processes
- » Acting as a peer coach for 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
- » Taking responsibility for successes and failures as a team
- » Taking the initiative to be self-managing
- » Being active and present throughout

Like organizational commitment, individual commitment is important beyond the agile transition period. The people on the first pilot will become a reference model as well as change agents throughout the company, setting the stage and exemplifying for other 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, structure, and mind-set changes.

Another important way to get commitment is to identify challenges with the organization's current development practices and provide potential solutions with agile approaches. Agile values, principles, and frameworks (such as scrum) can help address many problems, including issues with product quality, customer satisfaction, team morale, budget and schedule overruns, funding, portfolio management, and overall product issues.

Finally, highlight benefits of agile approaches to product development. 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 approaches often have higher customer satisfaction because scrum teams produce working products quickly, can respond to change, and collaborate with customers as partners.
- » **Profit benefits:** Agile approaches allow teams to deliver functionality to market quicker than with traditional approaches. Agile organizations can realize higher return on investment, often resulting in self-funded teams.
- » **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 product development in Chapter 21.

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 product progress?
- » **How are you doing business today?** How is product development 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 early learning from failure accepted? 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 product development was 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.
- » **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 but in a different way.

- » **Negative previous experience with attempted agile transformation.** Employees may have had a previous experience of a failed agile transformation and are turned off by the idea of trying it again. This situation can be an opportunity to return to agile values and principles. Proper training, an experienced agile mentor, and a focus on agile fundamentals are essential for transformation success.
- » **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 teams and functional managers. Demonstrate how the shift provides more effective outcomes for everyone. You can read more about servant leadership in Chapter 16.

Keep in mind that some resistance will arise; change can't happen without opposition. Be ready for resistance, but don't let it discourage you.

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 product development 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 whatever failed in your approach, and you'll gain a springboard for your first agile pilot.
- » **When your challenge is doing accurate budgeting:** Run your first agile development effort 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 development effort that will allow you to be more informed when planning next year's budget.
- » **When you're starting a new product development effort:** Moving to agile processes when you have a new objective to accomplish 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 success. Here are things to think about when choosing people for the different roles in your organization's first agile pilot.

The agile champion

At the beginning of an agile transition, the agile champion will be a key person in helping ensure that the team can succeed. This person should be able to effectively and quickly influence each level of the organization that affects the pilot 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 team as it goes through the steps to establish agile processes.
- » Acquire the team members necessary for success, both for the first pilot as well as for all teams 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 influence 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 scrum 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 scrum teams.



TIP

Although the agile transition team addresses organizational impediments in support of pilot teams, it makes changes to optimize not just a few local teams but the entire organizational system. This process is called *systems thinking*. The issues raised by the one pilot team inform the agile transition team of system-wide issues, and the transition team addresses those issues in ways that will benefit the entire organization.

The agile transition team should

- » Be committed to organizational success through the continuous support of pilot teams.
- » Establish a clear vision and roadmap for how the organization will become more 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 coach the rules of scrum).
- » Operate as a scrum team, holding all five scrum events and implementing all three scrum artifacts.

Figure 18-1 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.

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.

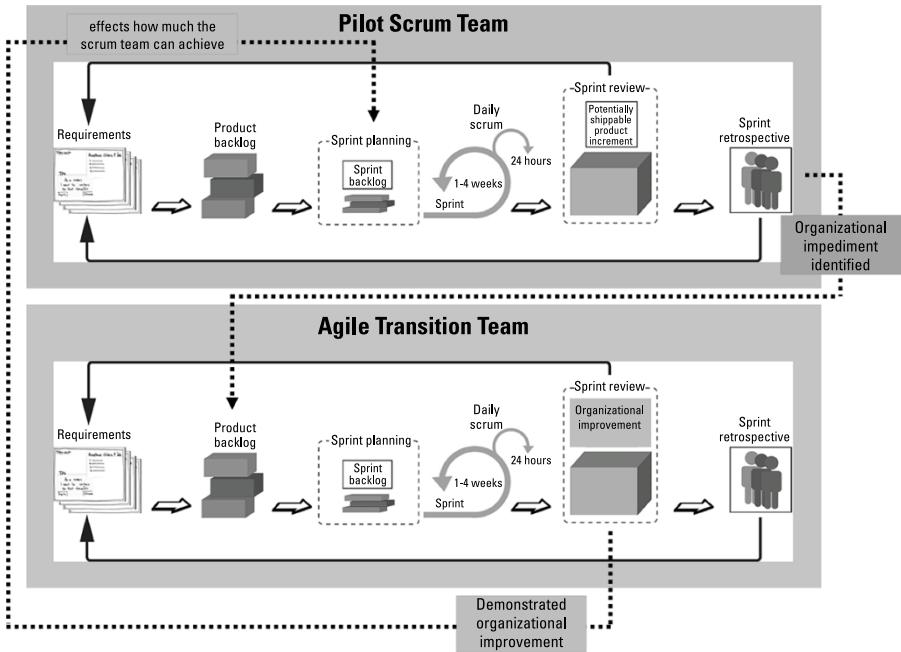


FIGURE 18-1:
Alignment of the
agile transition
team and the
pilot scrum team
cadences.

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 development effort, the product owner may need to acclimate to working on the product 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 development.
- » Have the ability to obtain product funding and other resources.

When choosing a product owner for your first agile pilot, find someone who can provide product expertise and commitment to the product. See Chapter 7 for more information about the product owner role.

The development team

With agile product development, the self-managing development team is central to the success of the product. The development team determines how to go about the work of accomplishing the product goals. Good development team members should be able to do the following:

- » Be versatile in skills.
- » 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 product goals in a variety of ways, are more likely to thrive in an agile environment. When choosing a development team for the pilot, 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. To learn more about the development team role, see Chapter 7.

The scrum master

The scrum master on a company's first agile product development effort may need to be more sensitive to potential development team distractions than on later efforts. A good scrum master should

- » Have influence (clout).
- » Have enough organizational influence to remove outside distractions that prevent the team from successfully using agile methods.
- » Know enough about agile product development to be able to help the team uphold agile processes throughout development.

- » 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 team, 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, challenge the status quo, and uphold agile processes in the face of organizational and individual resistance. See Chapter 7 for details on the scrum master role.

The stakeholders

On an organization's first agile pilot, good stakeholders should

- » Be involved.
- » Defer to the product owner for final product decisions.
- » Attend sprint reviews and provide product feedback.
- » Understand agile principles. Sending stakeholders to the same training as the rest of the team will help them be more comfortable with new approaches, processes, and techniques.
- » Receive product 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 team.

The stakeholders should be trustworthy, cooperative, and active contributors to a product.

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 (diversity in industry, business function, and role experience)
- » Be an expert at agile processes, especially in the agile approaches your organization chooses.

- » Be familiar with development efforts 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.
- » Guide the team through its first sprint at the beginning and be available to answer questions as needed throughout development.
- » 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 traditional 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 product development can be successful and teams can thrive. An agile environment refers to not only physical environments, such as the one we describe in Chapter 6, but also a good organizational environment. To create a good agile 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 18-2, using scrum and the other key agile practices to increase your chances of success. Start with the basics; build on them only when the product and your knowledge progress. Changing processes for the sake of perceived progress doesn't lead to perfection. Remember, practice doesn't make perfect; practice makes permanent. Start out correctly.
- » **Unfettered transparency:** Be open about development status and upcoming process changes. People on the team and throughout the organization should be privy to product development details.

- » **Frequent inspection:** Use the regular feedback loop opportunities that scrum provides to see firsthand how development is going.
- » **Immediate adaptation:** Follow up on inspection by making necessary changes for improvement throughout development. Take opportunities to improve today; don't wait until the end of a release or end of development.
- » **A dedicated scrum team:** The product owner, development team, and scrum master should be fully allocated to the development effort.
- » **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. Although collocation is preferred, consider equipment and training to make a virtual environment effective when collocation is not possible (as was the case during the COVID-19 pandemic, as discussed in Chapter 6).
- » **A well-trained product team:** When the members of the 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.

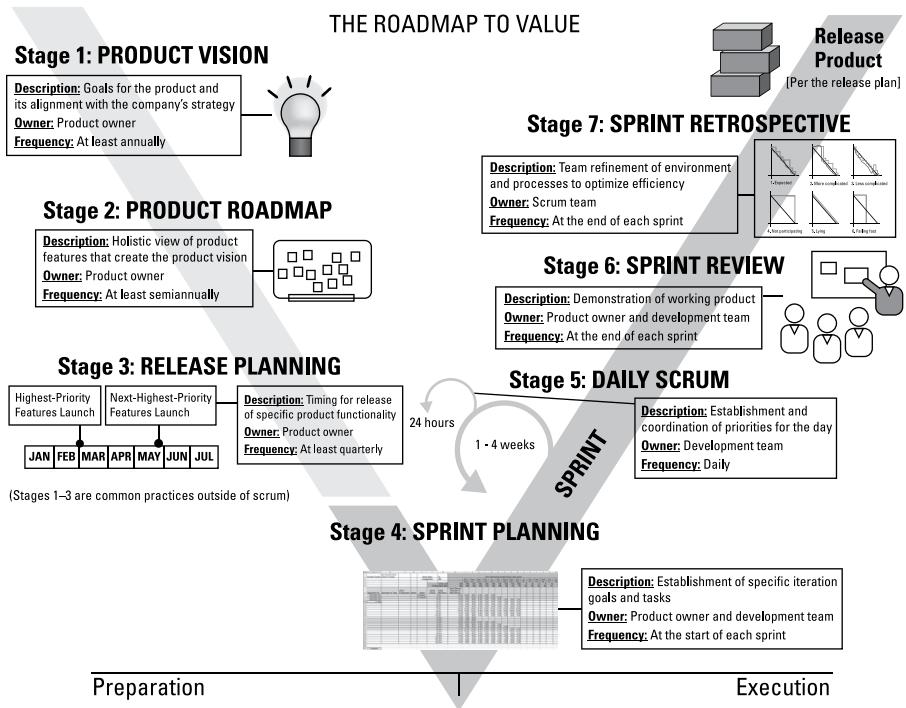


FIGURE 18-2:
The Roadmap to Value.

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)
 - Certified Scrum Professional (CSP) for ScrumMasters (CSP-SM), Product Owners (CSP-PO), and Developers (CSP)
 - Certified Team Coach (CTC)
 - Certified Enterprise Coach (CEC)
 - Certified Scrum Trainer (CST)
 - Certified Agile Leadership (CAL)
- » From Scrum.org:
 - Professional Scrum Master (PSM I, II, III)
 - Professional Scrum Product Owner (PSPO I, II, III)
 - Professional Scrum Developer (PSD I)
 - Professional Agile Leadership (PAL I)
- » From the International Consortium for Agile (ICAgile): Various professional and expert tracks in agile facilitation, coaching, engineering, training, business agility, delivery management, DevOps, enterprise, agility, and value management
- » From Kanban University
 - Team Kanban Practitioner (TKP)
 - Kanban Management Professional (KMP I, II)
 - Kanban Coaching Professional (KCP)
 - Accredited Kanban Consultant (AKC) and Trainer (AKT)

- » The Project Management Institute Agile Certified Practitioner (PMI-ACP) accreditation
- » 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 pilot that's important enough to get everyone's support. At the same time, set expectations: Although the pilot will produce measurable improvements, the results will be modest while the 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 principles at every level of the organization. Support 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 development efforts and teams. Also think about how you'll enable agile principles and techniques from a single product to many products, including those with multiple teams.

IN THIS CHAPTER

- » Identifying when and why to scale across multiple teams
- » Understanding the basics of scaling
- » Exploring scaling challenges

Chapter 19

De-Scaling across Teams

Depending on the schedule, scope, and required skills, many small and medium-sized product development efforts can be accomplished with a single scrum team. Larger efforts, 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. Regardless of the development effort's size, if interdependencies exist between multiple teams working together on the same product, or even across a collection of products, you may need to scale. Be aware, however, that scaling is an anti-pattern to agility.

With agile techniques, you decompose requirements into the most simple, independent channels of value, which enables the team to deliver working product early and continuously. Scaling is the opposite of decomposition, introducing dependencies. Rather than introducing overhead to deal with these dependencies, the goal is to simplify the breaking down of dependencies.



TIP

Scale only if you have to. Even though you may have the talent and resources available to enable multiple teams work on your product, 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 product 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 product and organization.

In this chapter, we discuss some of the issues to address when you need multiple teams for your product development effort. We also provide overviews of some common agile scaling frameworks and approaches that address the challenges of scaling.

Multi-Team Agile Development

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 product development, cross-functional teams work together during every sprint, 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 product include the following:

- » **Product 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 team (all scrum teams and stakeholders) and building a product roadmap with collaborative input from all parties involved requires a different approach than with a single-team product.
- » **Release planning:** Similar to the challenge of product planning, releases involve more specific planning of scope and release timing. Coordinating who will work on what and who will need help from whom throughout the release cycle is even more critical to ensure that dependencies, scope gaps, and talent allocation match the needs across the product.
- » **Decomposition:** To break down larger requirements in the same product 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, how can the scrum teams expose and address them at sprint planning?
- » **Daily coordination:** Even after effective planning and collaboration from 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 product 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.

Since the first scrum teams in the mid-1990s, there have been agile products requiring multiple scrum teams collaborating effectively. 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 a solution.



REMEMBER

The concept of vertical slicing applies to single team product development efforts, 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, touching all aspects of the technology and skills required.

Highly aligned and autonomous scrum teams do what needs to be done to meet customer needs. Development teams working on the same product are each cross-functional enough to work on any item from the product backlog. Each team pulls items from the same product backlog and implements on the same aligned cadence. If one team lacks a skill required for a requirement, they work with teams who do to expand their capability, reducing constraining dependencies.

During each sprint, each team integrates their work with the work of other teams, ideally through automation to make integrations less costly and troublesome. At the end of each sprint, all product backlog items selected during the sprint are elaborated, designed, developed, tested, integrated, documented, and approved.

Figure 19–1 illustrates how a product can be vertically sliced to enable multiple teams to work on a single product.

With a common baseline definition of done for all scrum teams, vertical slicing enables each team to be self-organizing in their development approach, yet aligned with the overall release goal. The product vision and roadmap become the glue keeping everyone together.

Scrum of scrums

Vertical slicing illustrates how scrum teams coordinate and integrate technically. How do these different scrum teams coordinate with each other daily? The *scrum of scrums* model is one way of facilitating effective integration, coordination, and collaboration among the people that make up scrum teams. Most scaling frameworks we show you in this chapter use scrum of scrums to enable daily coordination between scrum teams.

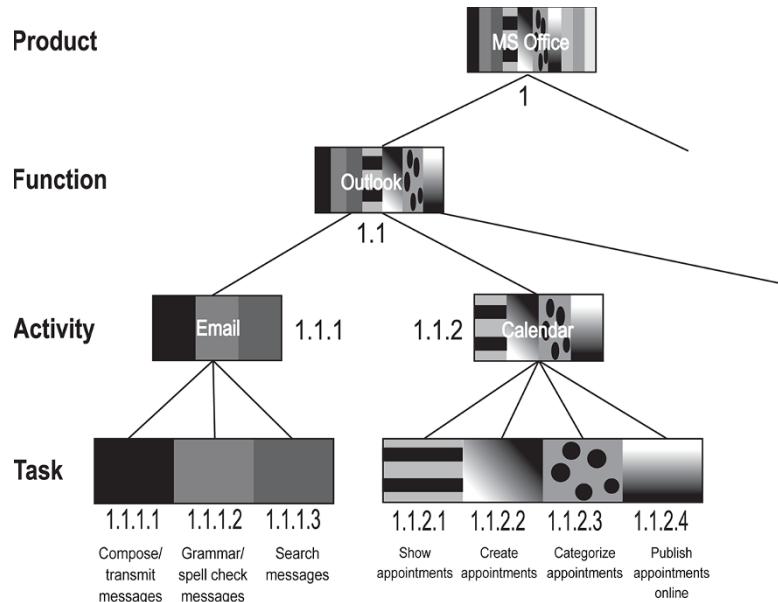
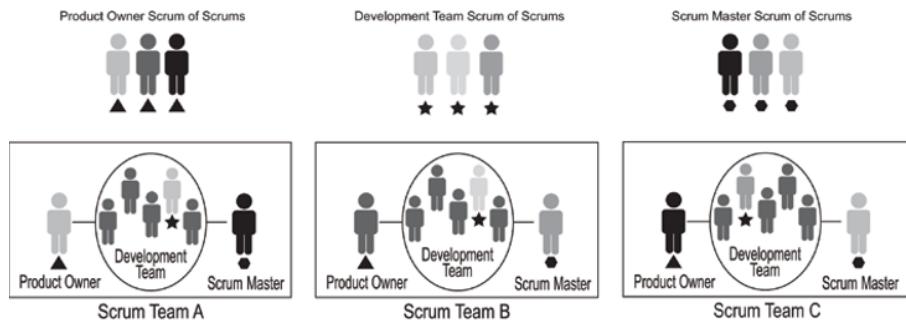


Figure 19-2 illustrates each role of each team coordinating daily with people of the same role in other teams regarding priorities, dependencies, and impediments that affect the broader program team. The scrum of scrums for each role is facilitated by someone designated and empowered by the group involved. Thorough integration and release efforts establish a consistent and regular scrum of scrums model.



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 or as often as needed, following the individual scrum teams' daily scrums, the product owners from each scrum team meet for no longer than 15 minutes. They address the priorities at play and make any adjustments based on the realities uncovered during the individual scrum team's daily scrum. Each product owner may address 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.")



TIP

In effect, the product owners form a band of aligned, self-organizing product owners and, similar to their own scrum teams, may huddle around a high-level summary view of what each scrum team is working on. Figure 19-3 is an example of what the product owners scrum of scrums board may look like.

	BACKLOG	DOING	DONE
Team 1	■ ■ ■	■	■
Team 2	■ ■	■	■ ■
Team 3	■ ■ ■	■	■ ■
Team 4	■ ■ ■	■	■
Team 5	■ ■	■	■ ■ ■

FIGURE 19-3:
Scrum of scrums
task board.

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 development teams scrum of scrums for no longer than 15 minutes to discuss the following:

- » Their team's accomplishments since the last time they met
- » Their team's planned accomplishments between now and the next meeting, and how their collective work will be integrated
- » 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, either daily or for each sprint, to ensure that everyone stays tuned in to the efforts of the overall product development.

Scrum master scrum of scrums

The scrum masters from each scrum team also meet with the other scrum masters 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

Impediments needing escalation are discussed and addressed after the daily scrum of scrums. Similar to the product owners and development team members, the scrum masters form their own band of aligned and self-organizing scrum masters.



REMEMBER

Your organization should have standards that guide and empower individual teams to make tactical decisions. This way, each team doesn't have to reinvent the wheel.

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.

Multi-Team Coordination with LeSS

Large-scale scrum (LeSS) is another way to scale scrum across massive product development efforts. 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 on system optimization so that each scrum team can be versatile in their ability to contribute effectively to the overall product backlog. It also presents a variety of options and approaches 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 19-4 illustrates the basic LeSS framework, using three scrum teams as an example. LeSS recommends no more than eight scrum teams follow the basic model.

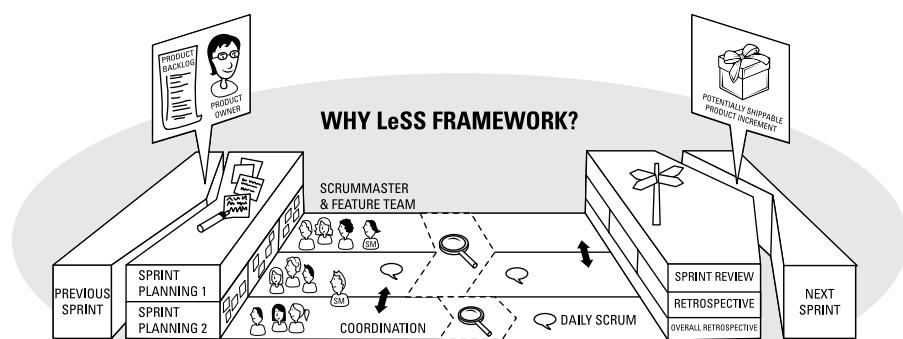


FIGURE 19-4:
Basic LeSS framework.

Used with permission, Craig Larman and Bas Vodde

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 retrospective. 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

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.

- » 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 product, such as processes, tools, and communication.

LeSS Huge framework

With LeSS Huge, a few thousand people could work on one product development effort. But the structure remains simple.

The scrum teams are grouped around major areas of customer requirements, called requirement areas. For each requirement 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). An overall product owner works with several area product owners, forming a product owner team for the product. Figure 19-5 illustrates LeSS Huge.

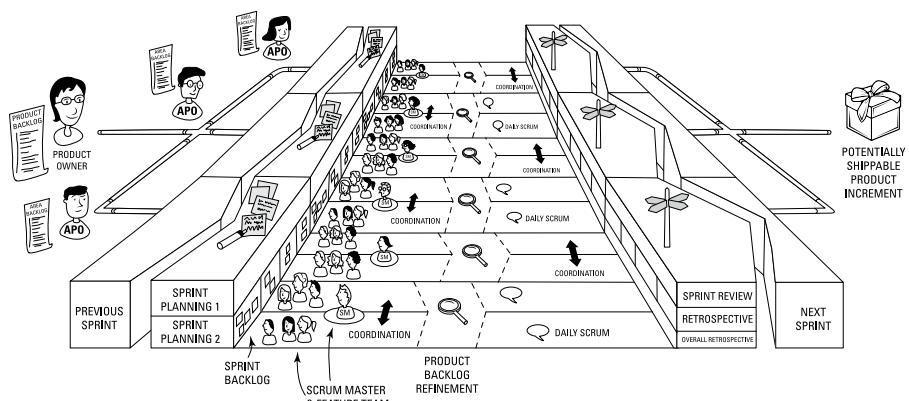


FIGURE 19-5:
The LeSS Huge
framework.

Used with permission, Craig Larman and Bas Vodde

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 product owner, and one sprint cadence across teams. LeSS Huge is simply a stacking of multiple parallel LeSS implementations for each requirement area.

To enable these teams to work together effectively across the requirement areas

- » The product owner regularly coordinates with the area product owners.
- » Requirement areas are flagged in 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 the product development effort.



REMEMBER

LeSS, just like single-team scrum, relies heavily on development teams with business domain knowledge and access to customers and business stakeholders empowered to elaborate requirements in collaboration directly with customers and the product owner. This collaboration is even more crucial in LeSS, where multiple teams share access with only one product owner.

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 5), LeSS suggests several options for scrum teams coordinating with other scrum teams, a few of them highlighted 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 a community 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.

A CoP 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 sessions, 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 5), 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. The traveler is careful to ensure that accountability for the product remains with the development team.

Aligning through Roles with Scrum@Scale

Agile scaling models vary in complexity and simplicity. The Scrum@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. The scrum of scrums model for the scrum master and product owner roles makes this daily synchronization possible between teams across programs of varying sizes.



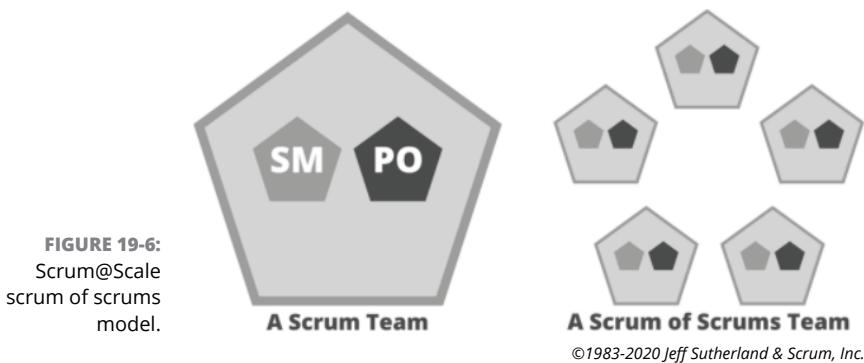
According to the Scrum@Scale guide, if an organization can't scrum, it can't scale. Building on a solid foundation of empowered, small, self-organizing teams, the agile values and principles, and scrum values and scrum itself enables the most effective scaling.

The scrum master cycle

Scrum@Scale groups at most five scrum teams into a scrum of scrums that work together to deliver a product. With more complex products requiring additional scrum teams, more than one scrum of scrums may be needed, with at most five scrum teams each. Daily representatives from each scrum team (at least the scrum masters) attend a scaled daily scrum meeting. It mirrors the daily scrum for individual scrum teams, inspecting the collective progress and surfacing and removing impediments.

With Scrum@Scale, reducing the number of people involved in a scrum of scrums limits the communication complexities for effective cross-team collaboration. It provides better visibility into what the scrum teams are working on and how their work might affect each other.

Figure 19–6 illustrates the Scrum@Scale scrum of scrums model.



With product development efforts of more than five scrum teams, Scrum@Scale implements a scrum of scrums of scrums, where a representative of each 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 19–7 illustrates the Scrum@Scale scrum of scrums of scrums model.

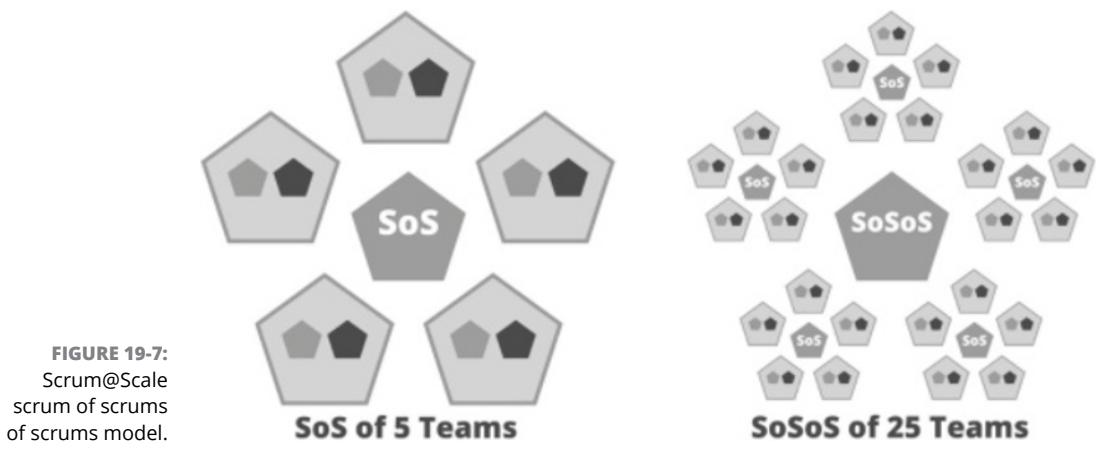


FIGURE 19-7:
Scrum@Scale
scrum of scrums
of scrums model.

©1983-2020 Jeff Sutherland & Scrum, Inc.

The executive action team (EAT), shown in Figure 19-8, fulfills the scrum master role for the entire organization, providing an agile ecosystem for enabling the scrum teams to function optimally. The focus of the EAT is ensuring that agile values and scrum are implemented effectively and the organization is optimized for it.

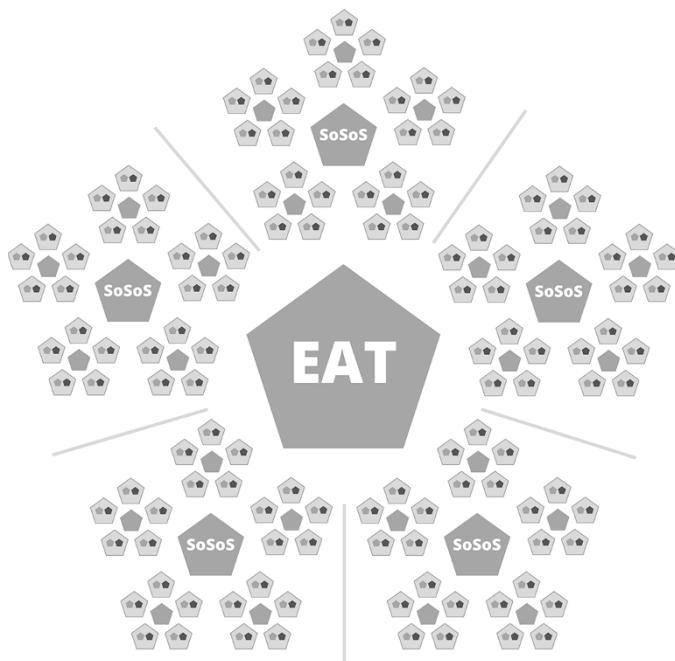


FIGURE 19-8:
Scrum@Scale
executive action
team (EAT).

©1983-2020 Jeff Sutherland & Scrum, Inc.

The product owner cycle

Each scrum of scrums has a common shared product backlog. The product owners organize in a similar and aligned way as the scrum masters in the scrum of scrums, only instead of labeling them as scrum of scrums, they are a product owner team.

A product owner team — including a chief product owner (CPO) and the product owners of each scrum team — is responsible for ordering the product backlog and ensuring that individual scrum teams align with the priorities. The CPO may be one of the individual product owners or someone else dedicated to it.

Each scrum team has a product owner and focuses on prioritization of their scrum team's sprint backlog. The product owner team communicates the overarching product vision.

Figure 19–9 illustrates the Scrum@Scale product owner team.

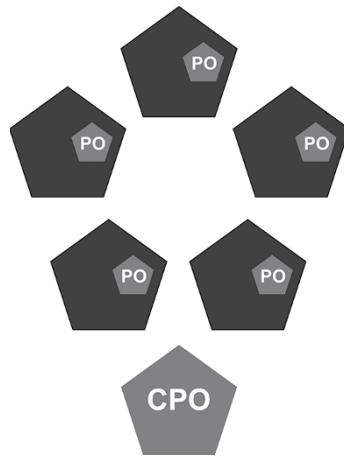
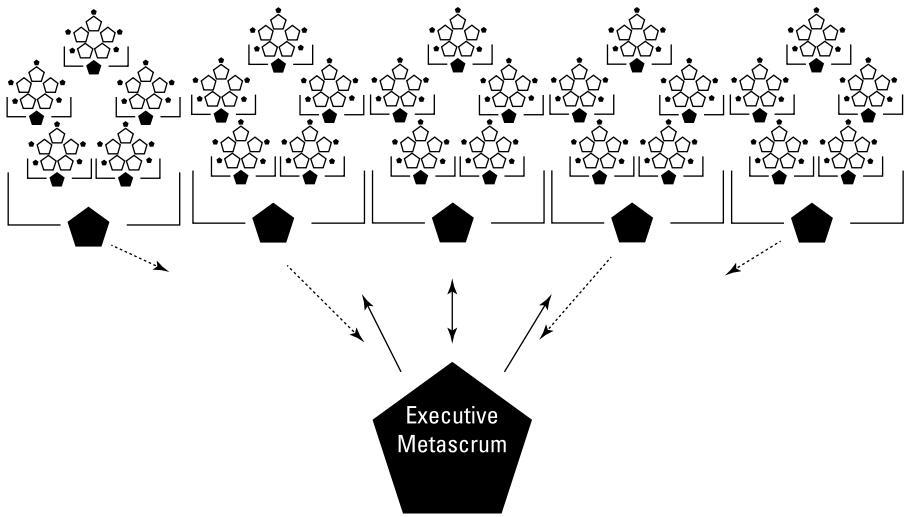


FIGURE 19-9:
Scrum@Scale
product owner
team.

©1983-2020 Jeff Sutherland & Scrum, Inc.

To coordinate product development for the entire organization, CPOs meet with executives and key stakeholders in an *executive metascrum (EMS)* event, which is the forum for collaborating and alignment on priorities, budget and maximizing delivery of value.

Figure 19–10 illustrates the Scrum@Scale executive metascrum (EMS).



©1983-2020 Jeff Sutherland & Scrum, Inc.

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 8:30 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 owner team associated with each scrum of scrums meet. At 9:00 a.m., if a scrum of scrums exists, scrum of scrums masters meet, and the product owners meet to coordinate priorities at the next level. Finally, at 9:15 a.m., EAT and EMS meet to coordinate and address any issues that need to be resolved for the scrum teams.

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. Figure 19-11 shows the full SAFe 5.0 big picture.

Organizations can choose from four SAFe configurations:

- » **Full SAFe** is the most complex configuration, supporting the building of large integrated solutions requiring hundreds of people or more. Refer to Figure 19-11.

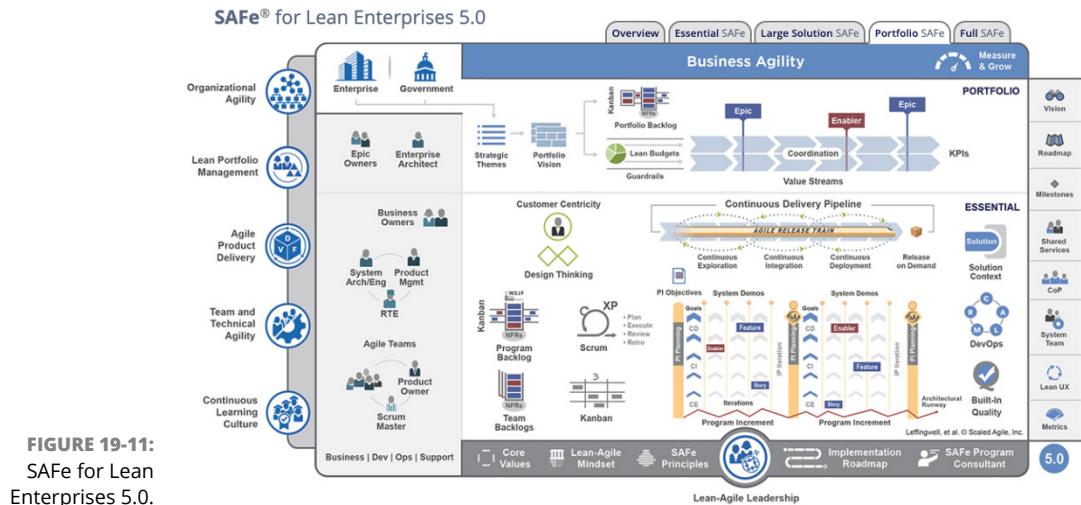


FIGURE 19-11:
SAFe for Lean
Enterprises 5.0.

Reproduced with permission from ©2011-2020 Scaled Agile, Inc. All rights reserved. SAFe and Scaled Agile Framework are registered trademarks of Scaled Agile, Inc.

- » **Portfolio SAFe** adds portfolio and investment funding, portfolio operations, and lean governance across the organization.
- » **Large Solution SAFe** is for building larger and more complex solutions but does not require portfolio-level constructs.
- » **Essential SAFe** is a basic starting point for smaller organizations and consists of the minimal elements necessary. See Figure 19-12.

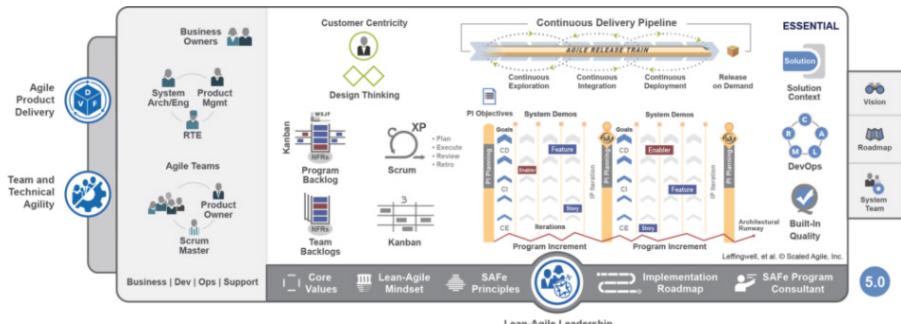


FIGURE 19-12:
Essential SAFe
configuration.

Reproduced with permission from ©2011-2020 Scaled Agile, Inc. All rights reserved. SAFe and Scaled Agile Framework are registered trademarks of Scaled Agile, Inc.

SAFe is focused around seven core competencies of the lean enterprise:

- » **Lean Agile leadership:** Leadership skills to drive and sustain organizational change by empowering people and teams to reach their potential
- » **Team and technical agility:** Driving team agile behaviors as well as sound technical practices including quality and development practices
- » **Agile product delivery:** Building high-performing collaborative teams of teams that use design thinking and customer focus to provide continuous flow of valuable products with various automation toolsets
- » **Enterprise solution delivery:** Building and sustaining the world's largest enterprise cyber-solutions, software products, and networks
- » **Lean portfolio management:** Executing portfolio vision and strategy formulation, chartering portfolios, prioritizing, and roadmapping
- » **Organizational agility:** Aligning strategy and execution by applying lean and systems thinking to strategy and investment funding, portfolio operations, and governance
- » **Continuous learning culture:** Continually increasing knowledge and competency and performance by becoming an organization committed to learning and innovating

Although other scaling frameworks have tactical differences, they also have many similarities, such as the following:

- » Development is done by development teams.
- » Teams are aligned in sprint length and cadence.
- » A scrum of scrums for coordination.

We don't go into all details of SAFe here, but we do highlight a few practices that address some of the scaling challenges discussed previously in this chapter.

SAFe introduces several new roles, titles, and structure names. In all SAFe configurations, you'll find at the team of teams level 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 ART provides a fixed cadence with which the teams align and synchronize. The rest of the organization, knowing this cadence, can also reliably plan its work around this known release train schedule.

ARTs are driven by a release train engineer (RTE), product management, and a system architect/engineer. The RTE acts as the scrum master of the release train.

Product management leadership provides product ownership guidance. The system architect/engineer provides technical leadership for the release train. The ART works at a cadence of five iterations by default to create what is known as a *program increment (PI)*, which is similar to the product increment concept in scrum.



WARNING

Scaled frameworks have the potential to allow your organization to remain unchanged in the spirit of reducing organizational disruption. The reality may be that the organization needs to change to better meet the needs of your customer. Choose from the scaled frameworks what makes the most sense for your organization.

Joint program increment planning

Joint program increment (PI) planning unifies agile teams across an ART. In PI planning, 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 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 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 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 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 the event. 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 up front, 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 16, 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 and large solution configurations 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 teams to be as effective and efficient as possible, as well as connect strategy to execution.



WARNING

If layers of leadership are added to your product development effort, be careful to protect the role of the product owner and his or her proximity to the customer. Many scaled teams have inadvertently fallen into the trap of disempowering their scrum teams.

Disciplined Agile Toolkit

Project Management Institute's (PMI) Disciplined Agile (DA) is based on the premise that true business agility comes from freedom, not frameworks. DA is a toolkit with hundreds of practices that guide teams and organizations to help improve the way they work.

Disciplined Agile is a learning-oriented hybrid approach to information technology (IT) solution delivery. It claims to be focused on delivering both risk and value, enterprise aware, and scalable.

Disciplined Agile's foundation layer outlines the principles (including agile and lean), guidelines, roles, and teams and how they work together.

Disciplined DevOps makes up the second layer where streamlining of software development and IT operations activities occurs. We discuss DevOps in Chapter 10.

Layer 3 is defined as value streams, based on the FLEX workflow, which tie the organization's strategies and enabling decisions for improving each part of the

organization in the context of the entire system. The goal is not just to be innovative but also to increase value realization through the following:

- » Research and development
- » Business operations
- » Portfolio management
- » Product management
- » Strategy
- » Governance
- » Sales and marketing
- » Continuous improvement

Disciplined Agile Enterprise (DAE), layer 4 of DA, is about the ability to sense and respond to changes in the marketplace through organizational culture and structure. The DAE layer focuses on the rest of the organization's enterprise activities that support its value streams, including

- » Enterprise architecture
- » People management
- » Asset management
- » Finance
- » Vendor management
- » Legal
- » IT
- » Transformation

Disciplined Agile is a hybrid toolkit that builds on the foundation of other methods and frameworks, such as DevOps, extreme programming (XP), scrum, SAFe, kanban, and others. The frameworks provide the process bricks while DA provides the mortar to hold the bricks together.

As mentioned at the beginning of this chapter, scaling is an anti-pattern. The best way to prevent the need for scaling is to enable your teams to become highly aligned and highly autonomous. Break down work and features into the smallest and most valuable increments.

Conway's Law (by Melvin Conway, who introduced the law in 1967) states that "Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations." Similarly, your product is a reflection of your organization, so organize as simply as possible.

Cultural changes such as the adoption of agile techniques require organization-wide commitment to long-term changes in mindset and structure, which we discuss next in Chapter 20.

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 principles
- » Leading change by example

Chapter **20**

Being a Change Agent

If you're contemplating the idea of introducing agile product development 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 product development techniques. We also introduce common change models, including the model we use at our company, Platinum Edge. We also cover leading change by example, as well as common pitfalls to avoid in your agile transition.

Becoming Agile Requires Change

Traditional approaches to project management focus significantly on processes, tools, comprehensive documentation, contract negotiation, and following a plan. Although agile product development 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 that may feel foreign.

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. David Rock developed a brain-based model for identifying five key domains that influence our behavior in social situations, often involving change. It's called the SCARF model, which stands for

- » Status: Our relative importance to those around us
- » Certainty: Our ability to predict what's coming
- » Autonomy: Our sense of control
- » Relatedness: How safe we feel around others
- » Fairness: How fair we perceive the exchanges between people

Each of these domains activates a threat or reward response in our brain, which is what we rely on for survival and is reflected in our behavior. You might be able to identify which of these domains influences you to feel threatened. Change is difficult for each of us, for a myriad of reasons.

We see these reactions firsthand as we help organizations make these changes. As consultants and trainers, 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. Excitement isn't enough for change.

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?
- » 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. Commitment doesn't just mean writing a check for the training and coaching. Commitment means that leaders start learning how to lead the change instead of outsourcing it to a consultant. It's an ongoing journey, not a destination.

Strategic Approaches to Implementing and Managing Change

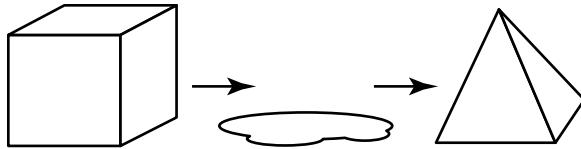
Organizational change initiatives typically fail without a strategy and discipline. Here, we define *failure* as not reaching the desired end state goal of what the organization will look like after the change. Failure is often due to being unclear as to the goal or because the change plan doesn't address 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 20-1.

FIGURE 20-1:
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. Typically, these steps are completed in the order listed, but a non-linear approach is realistic in our experience. You may need to readdress previous steps multiple times as you progress through each step.

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. Once desire exists for the individuals in an organization, there is motivation to move together to change.

Knowledge

Desire is key, but it won't result in change by itself. 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 often comes from expanding understanding and skills through 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 each step.

Kotter's eight steps for leading change

John Kotter's eight-step 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.

- » **Create a sense of urgency.** The leadership action is to create a sense of urgency to pull people out of complacency. 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.
- » **Build a 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.
- » **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 9 — change management also needs to begin with a clear vision of where you're headed.

- » **Enlist a volunteer army.** The leadership action is to enlist a volunteer army. Change will accelerate and last if massive numbers of 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.
- » **Enable action by removing barriers.** 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.
- » **Generate 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.
- » **Sustain acceleration.** 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.
- » **Institute change.** 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 successfully become an agile organization.

Figure 20-2 illustrates our agile transition roadmap for successful agile transformation.

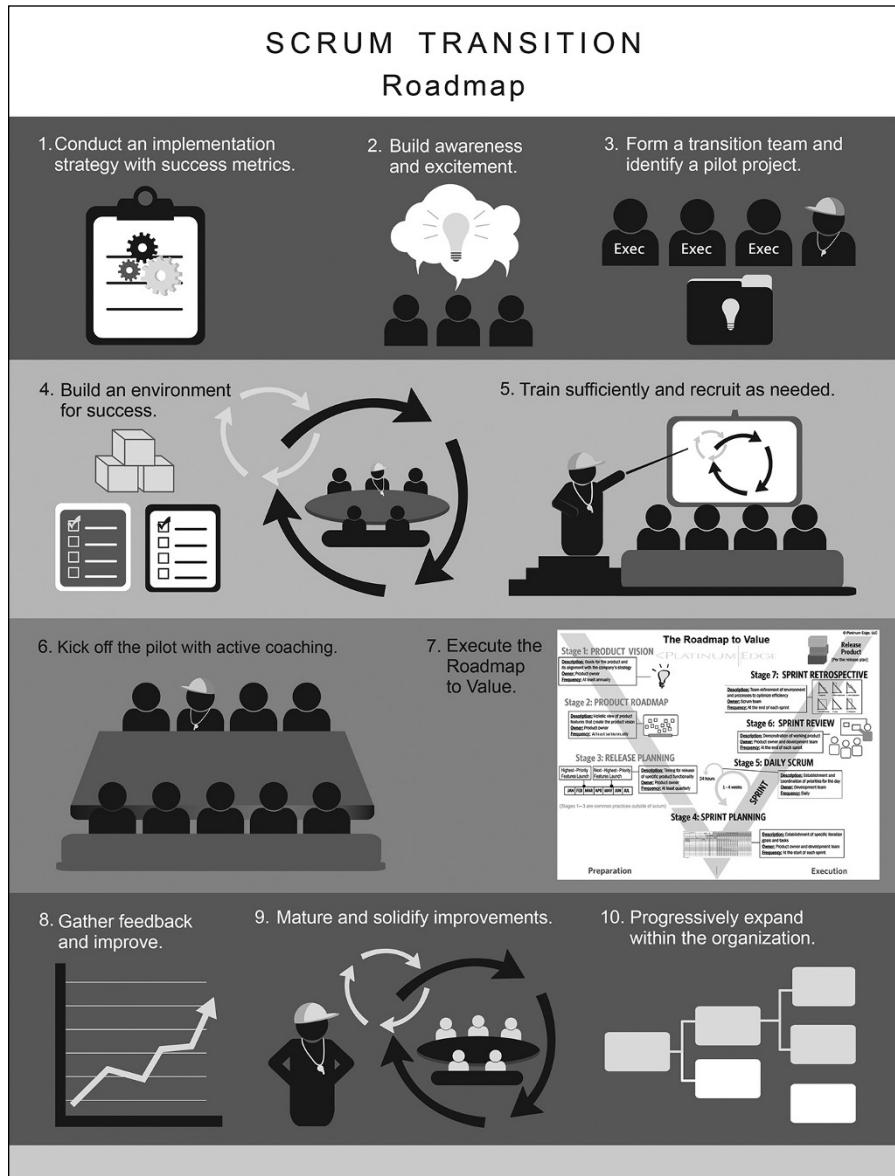


FIGURE 20-2:
Platinum Edge
agile transition
roadmap

Step 1: Conduct an agile audit to define an implementation strategy with success metrics

An *agile audit* of your organization is

- » A three- to five-week review of the existing project management, product development, corporate structure, objectives, and culture
- » Identification of opportunities to improve efficiency, effectiveness, and agility
- » Creation and presentation of an implementation strategy and roadmap

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 product development



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 develop products 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 training will be required? 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 product development effort, identify a quantifiable way to recognize success. Using metrics will give you a way to instantly demonstrate success to stakeholders and your organization. Metrics provide specific goals and talking points for sprint retrospectives and help set clear expectations for the 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 success measurements can do more than help you improve throughout your work. Metrics can provide clear proof of success when you move past your first product and start to scale agile practices throughout your organization.

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 product development. 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 21 has a great list of the benefits of agile product development 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 pilot know about the upcoming changes. Involve the pilot 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 the agile transition. 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.

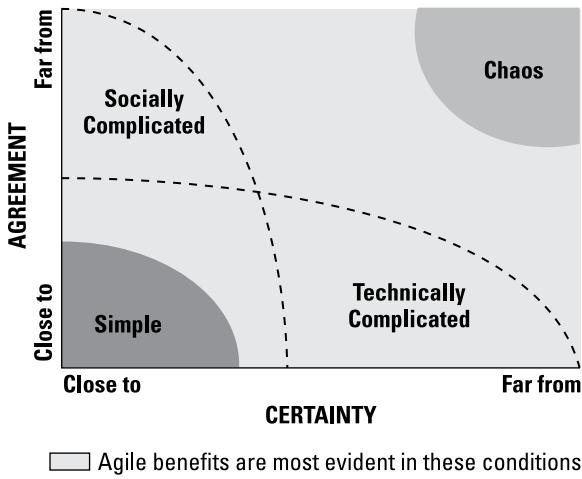
Step 3: Form a transformation team and identify a pilot

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 18, is made up of executives and other leaders who will systematically improve processes, reporting requirements, and performance measurements across the organization. Selecting people for the team who are passionate about and committed to helping the organization become more adaptive and resilient is paramount.

The agile transition team will create organizational changes within sprints, just like the development team creates product features within sprints. The transition 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 is a great way to establish a reference model of what a scrum team can look like and to demonstrate the benefits of an agile approach. Having a reference model allows you to figure out how to work with agile methods with little disruption to your organization's overall business. Concentrating on one pilot to start also lets you work out some of the kinks that inevitably follow change. Figure 20-3 shows the types of development efforts that benefit most from the agile approach.

FIGURE 20-3:
Product development efforts that can benefit from agile techniques.



When selecting your first agile pilot to establish a reference model for future scrum teams, look for an endeavor with these qualities:

- » **Appropriately important:** Make sure the product you choose is important enough to merit interest within your company. However, avoid the most important product coming up; you want room to make and learn from mistakes. See the note on the blame game in the later section "Avoiding Transformation Pitfalls."
- » **Sufficiently visible:** Your pilot 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 product development efforts 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 product that has a distinct end point, rather than one that can expand indefinitely.

- » **Not too large:** Select a pilot that you can complete with no more than two scrum teams working simultaneously to prevent too many moving parts at once. A single-team pilot is preferred.
- » **Tangibly measurable:** Choose an endeavor 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 20-4, illustrates the process of teams' excitement, chaos, and finally adjustment to new processes.

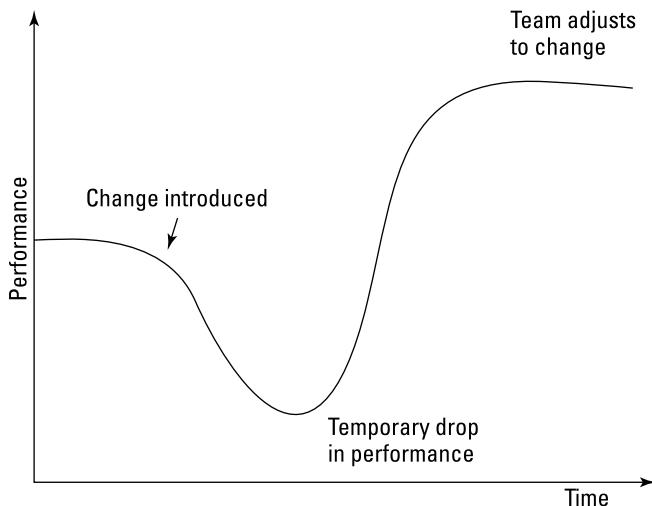


FIGURE 20-4:
Satir's Curve.

After you've successfully used agile techniques on one agile product development, you'll have a reference model and 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 6. Study the 4 agile values and the 12 agile principles carefully (see Chapter 2) and seriously determine whether you're creating an environment for success or rationalizing that the status quo is good enough.

Start fixing and improving your physical and cultural 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 team to absorb and develop the knowledge needed to successfully begin.

Training works best when the members of the team can train and learn together, and then bring the shared experience back to work with them. A common language and understanding exist among them. As agile trainers and mentors, we've had the opportunity to overhear conversations between 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 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 pilot 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 team, a pilot product 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 your pilot right.



TIP

Practice doesn't make perfect. Practice makes permanent. Start off right.

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 10) 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,** assuming sprint 1 was a success, 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. High-performance patterns — such as teams that finish early accelerate faster — can be learned earlier by using shorter sprints.



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 and especially if anyone is feeling stuck. 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 scrum 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 10, you can see how the scrum team can plan the sprint. Chapter 11 provides the day-to-day details on running the sprint.

Step 7: Execute the Roadmap to Value

When you've chosen your pilot, don't fall into the trap of using a plan from an old methodology or set of habits. Instead, use agile processes from the start.

We outline the Roadmap to Value throughout this book, introducing it in Chapter 9 and leading you through each of the seven stages in Chapters 9 through 12.

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. Stakeholders may initially be taken aback by a bare-bones approach. However, those stakeholders will soon be impressed as they find a working product increment 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 12, 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. (Focus on outcomes achieved over outputs produced.)
- » Go over how well you completed requirements to meet the definition of done: Designed, developed, tested, integrated, and documented.

- » Discuss how you met your success metrics or desired outcomes.
- » 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 12.

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” as introduced in Chapter 8. The term describes three stages in which people learn new skills:

- » **In the *Shu* stage**, new scrum teams may work closely with an agile coach or mentor to follow processes correctly.
- » **In 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**, scrum teams can customize processes, knowing what works in the spirit of the agile values and 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 stakeholders mature, entire companies can mature into successful agile organizations.

Step 10: Progressively expand within the organization

Completing a successful pilot is an important step in moving an organization to agile product development. With metrics that prove the success of your pilot and the value of agile methodologies, you can garner commitment from your company to support new opportunities for applying agile techniques.

To progressively expand the agile footprint across an organization, start with the following:

- » **Support new teams.** An scrum team that has reached maturity — the people who worked on the first agile pilot — should now have the expertise and enthusiasm to become agile ambassadors in the organization. These people can become part of a guild to help new teams to learn and grow. See Chapter 8 to learn more about guilds.
- » **Redefine metrics.** Identify measurements for success, across the organization, with each new scrum team and with each new product.
- » **Expand 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 19 for different ways of working across multiple teams.
- » **Identify new challenges.** Your first agile pilot 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 use agile techniques.

The preceding steps work for successful agile product development transitions. Use these steps and return to them as you expand, and you can enable agile principles to thrive in and drive your organization's success.

Leading by Example

Although the ten steps for successful agile transition are extremely helpful in agile transformations, an aspect that is imperative to every successful transformation is executive ownership. Consultants and coaches advise, but it is the organization that needs to do the work. Leaders have a responsibility to “walk the walk and talk the talk” of the new culture they’re trying to create. Agile leaders

understand that every eye is watching them always, and learning to be an agile and adaptive leader is crucial.

The role of a servant-leader in an agile organization

We refer to *servant leadership* frequently throughout this book. In agile transformations, leaders thoughts shift from asking “What’s wrong with this team?” to “How can I help?” Cultural shifts are more successful when people know that their leaders are alongside them, experiencing change and giving up old ways.

Instead of practicing command and control management tendencies, agile leaders set a clear vision but allow people to figure out how to accomplish that vision on their own. After people start sprinting toward the new vision, the leader is nearby giving encouragement and support and bulldozing blockers. Here are some keys for effective servant leadership.

High-performing teams are transparent — the pilot teams will work to make *everything* transparent. They do this so they can make necessary adjustments as part of their empirical process control. Some information, such as velocity, may seem like a manager’s opportunity for improving team performance. Avoid the trap! Velocity is not a goal to be pushed and the velocity number is meaningless as a productivity metric — team outcomes are what matters! (We discuss how to use velocity appropriately as a planning tool in Chapter 15.) As a leader, retrospective results shared by the team may provide insight for organizational impediments that you can remove. Serve the team by removing blockers impeding their ability to reach their full potential.



TIP

Teams thrive when they feel psychologically safe. Harvard’s Dr. Amy Edmondson defines psychological safety as the belief that you won’t be punished or humiliated for sharing your ideas, questions, and concerns or for admitting when you made a mistake. Studies show that psychological safety enables moderate risk-taking, speaking your mind, and creativity. Psychologically safe people are comfortable being themselves.

Keys for successful servant leadership

Following are keys for successful servant leadership:

- » **Build individual and organizational capability.** The more capable the teams become, the more effective they become. Teams get better at delivering value when individual team members get better at contributing to the

team's success. Invest in training so that individuals are able to contribute to their team's success in more than one way. Encourage team members to select tasks with which they may be unfamiliar so that they can stretch their skills and capability as they perform unfamiliar work. Build not only individual capability but also team capability. Build T-, pi-, and M-shaped individuals and pi- and M-shaped teams. (For more on T, pi, and M-shapes, see in Chapter 7.)

- » **Manage by exception.** Serve where you're needed. If a team is thriving, stretching, and accomplishing, stay out of their way (but continue to provide support). Focus your attention on the teams that need you most but, again, don't get in the way. Offer support by asking how you can help. Reviewing sprint burndown charts will provide a clear insight into progress and is an excellent opportunity to watch for trends. Burndown charts can highlight where you might be able to offer help. (See Chapter 11 to learn more about burndown charts.)
- » **Adjust the manager-to-creator ratio.** Agile organizations are self-organizing and self-managing. Rather than manager support, scrum teams need servant leadership support. The manager-to-creator ratio can be helpful for evaluating if you have the right ratio of managers to people developing your products. Agile organizations have low manager-to-creator ratios because the creators are empowered to do what they do well, and the managers redirect their time from micromanaging to providing the support and trust creators need. The organization will have more empowered team members supported by servant-leaders.

Avoiding Transformation Pitfalls

Organizations can make a number of common but serious mistakes when implementing agile practices. Table 20-1 provides an overview of some typical problems and ways to turn them around.

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 team is trained, and if the scrum team makes an active commitment to upholding agile values, you'll have a successful agile transition.

TABLE 20-1

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 development efforts, 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. See Chapter 23 to learn more about faux or fake agile.</p> <p>Trying to complete agile processes in addition to waterfall processes, documents, and meetings is double the work. <i>Double work agile</i> results in quick team burnout. If you’re doing twice the work, you aren’t adhering to agile principles.</p>	Insist on following one process — an agile process. Garner support from management to avoid non-agile principles and practices.
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>	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.
Ineffective product owner	The product owner role is non-traditional. Scrum 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 product development effort.	<p>Start the pilot 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 on how to leverage agile principles and approaches, such as scrum, to build great products, 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 who can make product decisions and help the scrum team be successful.</p>
Lack of automated testing	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.	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.

(continued)

TABLE 20-1 (continued)

Problem	Description	Potential Solution
Lack of transition support	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.	When you decide to move to agile product development, enlist the help of an agile mentor — ideally externally with a diverse set of experience and impartial perspective — who can support your transition. 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.
Inappropriate physical environment	When scrum teams are not collocated, or otherwise enabled to collaborate in real-time and high fidelity, they lose the advantage of face-to-face communication. Scrum teams work best when they can sit together in the same area or, if distributed, in the same time zone.	If your scrum team is in the same building but not sitting in the same area, move the team together. Consider creating a room or annex for the scrum team to continually collaborate. 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. Before starting a pilot 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 Chapter 16 to see how to manage dislocated teams.
Poor team selection	Scrum team members who don't support agile processes, don't work well with others, or don't have capacity for self-management will sabotage a new agile pilot from within.	When creating a scrum team, consider how well potential team members will enact the agile principles. The keys are versatility and a willingness to learn.
Discipline slips	Remember that agile products still need requirements, design, development, testing, and releases. Doing that work in sprints requires discipline to keep from falling into old habits of, for instance, delaying testing until the end of the sprint.	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 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 agile values and principles. Resist changes that aren't compatible. Remember that maximizing work not done is essential.

Avoiding agile leadership pitfalls

Although there are pitfalls to avoid from an organizational transformation perspective, here are a few tips to enable organizational leader success:

- » **Hold the right people accountable.** Many leaders hold their direct reports accountable for product delivery. In an agile organization, accountability for what is built and when it's delivered to customers lies with the product owner, and accountability for building quality into the product lies with the development team. Organizational leaders are servant-leaders helping product owners make better decisions and creating opportunities for building capable development teams who can build products to meet customer needs. Having an organizational leader become accountable for something a product owner or development team owns leads to frustration and de-motivation.
- » **Be open to what could be.** Many leaders, especially new leaders to the organization, believe the flavor of their agile at their old organization was better than the agile practices used in their new organization. Begin where your new organization is and learn why they develop products the way they do. Perhaps you can sign up to attend the same training received by your teams to better align with the organization's training. Beware of inadvertently undermining the organization's agile trajectory.
- » **Get trained.** The most effective transformations begin with leaders who lead by example. Getting trained is the first and most important step leaders can take to start exemplifying the agile culture they want to create. Otherwise, pilot teams will wonder why they are being asked to practice agile techniques when their untrained leaders don't. Avoid this trap by investing time and energy into getting trained.
- » **Inspect and adapt.** Use the sprint review as an opportunity to not only show your support for the team but also to inspect and adapt. Many coaching opportunities become exposed during a sprint review. Your absence from a sprint review sends a message that you believe whatever the team is working on is not important. Take advantage of the "olympic stage" performance improvement opportunity discussed in Chapter 12.
- » **Practice agility yourself.** Gather your leadership team around you and use scrum. Build a backlog and make it transparent to your organization. Plan your sprint, hold a daily standup, demonstrate your work to your organization in a sprint review, and then hold a team retrospective. If you think scrum is a good idea for your organization, isn't it a good idea for you too? Teams who see their leaders using scrum will be more motivated to improve their scrum practice.

» **Trust others.** Scrum teams fail early and often. Failure is essential for learning. Avoid helicopter-managing and allow teams to solve their own problems as much as possible. As a leader, you know there is no substitute for hard-fought, lesson-learned experience. Let natural consequences lead to unforgettable learning opportunities. Many traditional management tendencies may need to be unlearned to help teams benefit from autonomy, mastery, and purpose, as discussed in Chapter 8.

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:



REMEMBER

- » Are you doing “scrum, but . . .”?

“ScrumBut” occurs when organizations partially adopt scrum. Beware of old practices that thwart agile principles, such as finishing sprints with incomplete functionality.

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 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. The best way to learn how a team is performing is to attend their sprint review. You can see more about story points and velocity in Chapter 10 and sprint reviews in Chapter 12.

- » 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. Scrum teams we work with start developing as early as day two.



TIP

» 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.

Offshoring is common. We're all for it, but when you offshore, offshore with both feet. Create scrum teams in single geographic locations so they can be collocated or at least collaborate all day in similar time zones. Having a fully collocated scrum team on one continent and another fully collocated scrum team on another continent is recommended.

If you do work with offshore teams, invest in good collaboration tools such as individual video cameras and persistent, virtual team rooms. Short sprints enable frequent inspection and adaptation opportunities, which is particularly helpful for offshore and other vendor engagements.

» 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 product development 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 — there is no boss on 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.

The Part of Tens

IN THIS PART . . .

Communicate the benefits of agile product development.

Address key factors for agile success.

Common signs and indications that you're *not* agile.

Become an agile professional through learning, networking, and community collaboration, with support from valuable resources.

IN THIS CHAPTER

- » Ensuring that products are rewarding
- » Making reporting easy
- » Improving results
- » Reducing risk

Chapter **21**

Ten Key Benefits of Agile Product Development

In this chapter, we provide ten important benefits that agile approaches provide to organizations, scrum teams, and products.



REMEMBER

To take advantage of agile product development benefits, you need to trust in agile principles, learn more about different agile practices and approaches, and discover what's best for your team.

Higher Customer Satisfaction

Scrum teams are committed to producing products that satisfy customers. Agile approaches for happier customers include the following:

- » Collaborate with customers to collect feedback throughout the process so customers get what they really want.
- » Ensure that the product owner is an expert on product requirements and customer needs, or knows where to get that information. (Check out Chapters 7 and 11 for 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 10 and its role in responding to change in Chapter 14.)
- » Demonstrate working functionality to stakeholders in every sprint review. (Chapter 12 shows you how to conduct a sprint review.)
- » Deliver products to market quicker and more often with every release.

Better Product Quality

Customers demand quality products. Agile methods have excellent safeguards to make sure that quality is as high as possible. Scrum 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 thorough testing into the development process, allowing the development team to address issues while they're fresh.
- » Take advantage of automated testing tools, ensuring that new product increments do not break previous increments.
- » 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 agile quality in Chapter 17.

Reduced Risk

Agile product development techniques virtually eliminate the chance of absolute failure — spending large amounts of time and money with no return on investment. Scrum teams reduce risk by doing the following:

- » Develop in sprints, ensuring a short time between initial investment and either failing fast or knowing that a product or an approach will work.
Front-load risk by ensuring that the product backlog is stacked with the most valuable and most risky items first. See Chapter 13 to find out about assessing product risks and opportunities.
- » Always have a working, integrated product, starting with the first sprint, so that some value is added as shippable functionality every sprint, ensuring the product won't fail completely.
- » Develop requirements according to the definition of done in each sprint so that sponsors have completed, usable functionality, regardless of what may happen with the product 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 products, allowing organizations to pay for a product with little up-front expense.

You can find more information about managing risk in Chapter 17.

Increased Collaboration and Ownership

When development teams take responsibility for products, they can produce great results. Agile development teams collaborate and take ownership of product quality and 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, roadblocks, and team morale.
- » 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. If you're on a distributed team, stay connected to your team's videoconference.
- » 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 9. To discover more about sprint planning and daily scrum meetings, see Chapter 11. For more information about sprint reviews and retrospectives, check out Chapter 12.

More Relevant Metrics

The metrics that scrum teams use to estimate time and cost, measure performance, and make 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. With agile product development, you provide metrics by doing the following:

- » Determine 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 requirements.
- » Use relative estimates, rather than absolute 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 product.
- » 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 teams determine when to end development and redeploy capital to a new investment opportunity.



WARNING

You might notice that velocity is missing from this list. *Velocity* (a measure of development speed, as detailed in Chapter 15) 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 9. You can find out about tools for determining timelines and budgets, along with information about capital redeployment, in Chapter 15.

Improved Performance Visibility

With agile product development, every member of the team has the opportunity to know how the product development is going at any given time. Teams 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 product.
- » 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 scrum team may speak at the daily scrum meeting, any member of the product team may observe or listen.
- » 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 product development visibility can lead to greater investment control and predictability, as described in the following sections.

Increased Investment Control

Scrum teams have numerous opportunities to control investment performance and make corrections as needed because of the following:

- » Adjusting priorities throughout development allows the organization to have fixed-time and fixed-price products while accommodating change.
- » Embracing change allows the 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 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 processes.
- » Self-organizing, self-managing teams possess the potential for self-funding products. (Chapter 15 tells you about self-funding products.)

The many opportunities to inspect and adapt throughout agile product development allow all members of the product team — the product owner, development team, scrum master, and stakeholders — to exercise control and ultimately create better products.

Improved Predictability

Agile product development techniques help the team accurately predict how things will go as product development progresses. Here are some practices, artifacts, and tools for improved predictability:

- » Keeping sprint lengths and development team allocation the same throughout development allows the team to know the exact cost for each sprint.
- » Using individual development team speed allows the 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 team to predict performance for individual sprints.

You can find more information about sprint lengths in Chapter 10.

Optimized 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 product development efforts can have multiple scrum teams, if necessary. Self-management and size-limiting mean that agile product development can provide unique opportunities to customize team structures and work environments. Here are a few examples:

- » 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.
- » Because development teams estimate the work they will do, product owners can determine how many scrum teams may be required to accomplish the items on a product backlog.
- » 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 product development 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 product development 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. People benefit from improved autonomy, mastery, purpose, and belongingness.
- » 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 16.

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

Chapter **22**

Ten Key Factors for Agile Product Development Success

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.



We have found that the first three are the strongest indicators for success. Get those right and the likelihood of your success increases dramatically.

TIP

Dedicated Team Members

In Chapter 8, we describe how products are considered long-term assets requiring stable, dedicated, and even permanent teams. Permanent teams retain knowledge about their product and customers. Their high performance is built over years of retrospectives and hard work. Minor adjustments to the team may need to be made due to career opportunities and the like, but for the most part organizations should work to disrupt team makeup as little as possible.

Further, it's critical for these dedicated team members — product owner, development team members, scrum master — to focus on a single objective at a time. If team members are jumping between contexts hourly, daily, weekly, or even monthly, their effectiveness is minimized due to the increased cost of just trying to keep up with multiple task lists. The time lost due to the continual cognitive demobilization and remobilization involved with task switching is costly.



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 priorities simultaneously. The American Psychological Association reports that task switching wastes as much as 40 percent of time.

Variance in equals variance out.

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 development.

Challenges associated with dislocated teams include the tendency to rely on written communication due to the inconvenience of communicating face-to-face across geographic or even time zone separations. Written communications, especially those intended to resolve issues, are prone to costly delays and misunderstandings. The lack of real-time, face-to-face interactions results in lower levels of trust and familiarity with how other people think, act, and work.

In Chapter 6, we talk about collocation as the first crucial element of an agile environment. Bell Laboratories showed a fifty-fold improvement in productivity. One of the important factors in their success was collocating, which led to increased customer collaboration, working functionality, and response time. Technologies such as videoconferencing and other digital collaboration tools make dislocated collaboration more effective. However, although technology helps compensate for the challenges of dislocation, it's not as valuable or sustainable as physically working next to your teammates. See Chapter 6 to learn more about creating an environment for team success.

Done Means Shippable

Ending sprints with non-shippable functionality is an anti-pattern to becoming more agile. *Done* means shippable. A sprint that ends without potentially shippable functionality is, by definition, not a sprint.

Development teams get to done by swarming on user stories — working together on one user story 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, including test automation, 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 10) before approving and moving on to a new user story.

Address What Scrum Exposes

Scrum won't solve any problems for you, but it will expose them. Scrum will expose weaknesses and gaps in process, policies, organizational structures, skill-sets, roles, artifacts, meeting effectiveness, transparency, and myriad other topics. What you decide to do with what is exposed is up to you. Scrum provides an iterative inspect and adapt framework for addressing items as they're exposed. Scrum teams should address what is exposed on their own when they have control over the exposed issue. When they don't have control over the exposed issue, there should be a path for escalating those items to those in the organization who own the status quo and who can affect change to support scrum teams getting better and better at delivering customer value. An effective way to resolve escalated items exposed by scrum is to use the agile transition team, as explained in Chapter 20.

Clear Product Vision and Roadmap

Although the product owner owns the product vision and product roadmap, many people affect the clarity of these agile artifacts. Product owners need access to and strong working relationships with stakeholders and customers throughout product development to ensure that the vision and roadmap continually reflect what the customer and market need. The development team must also be crystal clear on the purpose of everything it works on, from the product vision to the individual user story. 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 9.

Product Owner Empowerment

The product owner’s role is to optimize the value produced by the development team. This product owner responsibility requires that someone be knowledgeable about the product and customer, available to the development team throughout each day, and empowered to make priority decisions and give clarification immediately so that development teams don’t wait or make inappropriate decisions about the product’s direction.

Organizations with the following product owner characteristics will struggle significantly to deliver valuable and shippable functionality at the end of every sprint:

- » The product owner struggles to make tough business decisions.
- » The product owner is not accessible to the development team because he or she has too many other things to do besides support the development team and work directly with customers and stakeholders.
- » More than one product owner is named by the organization for a single product, confusing the development team as to whom to go to for clarification.
- » Stakeholders undermine decisions made by the product owner.

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 7 for more on the product owner role.

Developer Versatility

You probably won’t start your first agile product development effort with a development team that has the ideal level of skills required for every item in 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 in any one skill, including testing.

From day one, you need people 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 to get things to done as quickly as possible. This versatility is discussed more in Chapter 7.

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 a socially earned ability to influence, they can best serve their teams to optimize their working environment. In Chapter 7, 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.

Leadership Support for Learning

When organizational 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 needed to implement the changes. Realistically, organizational transformations take from one to three years from the time leadership buys in. Buying in means much more than writing the check for the training or coaching services. It means leaders get involved and learn what they need to do to lead the transformation from within, buying in with their time and effort and actions.

It is unrealistic to expect all the benefits of following agile principles after the first sprint. In Chapter 20, we talk about choosing an appropriate agile pilot, 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 20 compares an agile transition to a sports team learning 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
- » Reinforcing 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 20 for specific steps to take alongside your trusted agile expert coaches.

IN THIS CHAPTER

- » Understanding the organizational benefits from adopting agile techniques
- » Recognizing potential signs preventing agility
- » Watching out for faux agile

Chapter **23**

Ten Signs That You're Not Agile

The journey to become agile is popular, but business agility in itself is not the goal. It's a means to an end. Those who are making the journey, according to the Scrum Alliance, report the following benefits:

- » Faster time to market
- » More alignment with the business
- » Better visibility of product development endeavors
- » Increased ability to manage changing requirements

The journey to become agile is never ending, but the agile values and principles will guide you. Along the way, you might encounter some signs that your organization is not becoming more agile, as described in this chapter.

A Non-Shippable Sprint Product Increment

The first sign is easy to spot. If your product team is not able to demonstrate a potentially shippable product increment at the end of the sprint, you're not doing it right. Agile product development builds working product increments into each

and every sprint. The product owner may choose not to release the product increment, but it is potentially shippable nonetheless. Agile product teams adhere to their definition of done to instill confidence that they've built the right product increment in the right way. Not having a potentially shippable product increment at the end of a sprint is a sure sign that the team is struggling and may need help.

Signs to watch for are demonstrations of the product in non-production-like environments, a reluctance to let people use the increment, and focusing on attention-distracting PowerPoint presentations rather than working products. Teams in this situation often suggest that they need more time by extending their sprint length, which is an anti-pattern.

Agile values and principles for navigating this situation follow:

- » Value #2: Working software over comprehensive documentation.
- » Principle #7: Working software is the primary measure of progress.

Scrum teams value working products and demonstrate valuable increments during each sprint review.

Long Release Cycles

The longer your release cycles, the less likely you're becoming more agile. Long release cycles are typically the result of traditional waterfall practices, in which all the risk and value is saved until the end and then released. Or it can be a symptom of feeling the need to deliver everything before delivering anything. *Short* means different durations to different organizations and industries, but the principle of shorter being better than longer is the constant.



TIP

Long release cycles can be reduced by focusing on the next minimum viable product (MVP). Speed-to-market advantage is indefensible because first-to-market winners

- » Get the opportunity to be first to capture market share
- » Validate the viability of their product by being first to capture end customer feedback lowering the cost of pivot
- » Receive an early return on investment (ROI) realizing a dollar today is more valuable than a dollar in the future (net present value 101)
- » Avoid internal or external obsolescence

Agile product development releases early and often. Short feedback cycles provide learning for the team, helping them to incrementally align the product to their customer's needs and reduce risk.

If a team is unable to have short release cycles, here are some warning signs: product owners hesitant to release anything until they have everything, the development team lacking confidence in the quality of its work, an engagement with a vendor requiring a traditional waterfall approach, and most frequently, the phrase, “We can’t release until the other pieces of the product are complete.”

Scrum teams tackle the most risky and valuable requirements first. Early failure can be a form of success because of the speed of learning. You want to validate assumptions and vet good ideas early. Most organization we work with use one-week sprints. Customers demand more sooner.

Agile principles for navigating this situation follow:

- » Principle #1: Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- » Principle #3: Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

The key words are *early, continuous, valuable product, frequently, and shorter timescale*.

Disengaged Stakeholders

A key sign you’re not agile is when stakeholders or sponsors are disengaged. Stakeholder feedback is essential for ensuring that the product will meet the needs of the customers. Lack of stakeholder feedback puts the team in a precarious situation, essentially sailing a ship without a rudder.

Signs your stakeholders are disengaged are a lack of stakeholder participation during product vision and road-mapping sessions, release planning, and most obviously during sprint reviews. The inspection opportunity provided during sprint reviews is the stakeholders’ best opportunity to provide feedback.

Stakeholder engagement is a key responsibility of the product owner. We also encourage scrum masters to “never lunch alone,” continuously building relationships and influence throughout the organization to enable effective and timely removal of impediments.

Agile values and principles for navigating this situation follow:

- » Value #3: Customer collaboration over contract negotiation.
- » Principle #8: Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- » Principle #5: Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

Scrum teams who maintain a sustainable pace as they collaborate frequently with sponsors and stakeholders are more effective. Sponsors and stakeholders who give the teams the needed environment and support, and then trust the team to get the job done will see teams become even more motivated to build better products faster.

Lack of Customer Contact

As stated, the relationship between the scrum team and the stakeholders is essential, but another sign you're not agile is when an invisible wall is placed between the development team and the stakeholders or customers. Some organizations make the mistake of distancing the technical team from the business partners. Critical feedback needed by the team is missed when this occurs.



REMEMBER

When a development team gets its information indirectly from secondary sources, the process becomes much like the children's telephone game. The message changes each time one child whispers what he or she heard to the next child, so that when the child who started the chain hears what the last child heard, the message has become totally different. Similarly, customer feedback can get twisted and filtered, losing its meaning and intent.

The closer development teams can be to the customers and the more frequent their interactions, the better they'll understand the customer's problems and challenges. The development team's work comes to life and the motivation for solving problems for real people in real situations makes their work more meaningful.

Agile values and principles for navigating this situation follow:

- » Value #1: Individuals and interactions over processes and tools.
- » Principle #2: Welcome changing requirements, even late in development.
Agile processes harness change for the customer's competitive advantage.
- » Principle #4: Business people and developers must work together daily throughout the project.

Accurately capturing customer needs occurs when scrum teams focus on individuals and interactions, welcome change, and then harness that change for their customer's competitive advantage and work with business people daily.

Lack of Skill Versatility

Teams with limited skill versatility will become victims of dependencies and constraints. Mature scrum teams (those who have worked together for a long time and have self-organized effectively over time to increase skill diversity across the team) 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. (See Chapter 7 to learn more about eliminating single points of failure.)

Over time, as each person increases his or her quantity and level of skills, the constraints and delays due to skill gaps disappear. Scrum 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. Working on a task that requires a skill you haven't yet mastered is the first step to organically increasing the team's capability. Outside coaches with the necessary skills who join the team for short periods until the team learns the new skills can also help. Build M-shaped individuals and M-shaped teams, as discussed in Chapters 7 and 16.

Agile principles for navigating this situation follow:

- » Principle #9: Continuous attention to technical excellence and good design enhances agility.
- » Principle #11: The best architectures, requirements, and designs emerge from self-organizing teams.

Self-organizing, cross-functional teams who strive for technical excellence build both individual and team capability.

Automatable Processes Remain Manual

Avoidable product defects are another sign of impeded agility. Defects are generally the result of poor testing. Teams who manually test take longer and are less thorough. Manual testing doesn't keep up with the pace of change required in today's environment, and falls short of instilling confidence that everything still works whenever a change is made.

Lack of automation is often another barrier to releasing product increments timely. With software development, continuous integration and continuous deployment (CI/CD) pipelines enable teams to not only automate testing but also automate deployment scripts.

Including test automation and deployment automation in the definition of done is critical for becoming agile. Without automation, scrum teams will not be able to deliver early and often and respond quickly to changes in technology and the marketplace.

Agile values and principles for navigating this situation follow:

- » Value #4: Responding to change over following a plan.
- » Principle #3: Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- » Principle #9: Continuous attention to technical excellence and good design enhances agility.

Prioritizing Tools over the Work

Although tools (including digital tools) can be helpful to scrum teams, they can also cause unnecessary overhead burdens and anti-patterns. A sign that you're not agile is when maintenance of the team's tool requires more time and effort than the development of the product itself. When the tool becomes the team's focus rather than the product they're building, you're not agile.



TIP

When shopping for tools to enable the product development work, remember that a tool that takes a minute a day to update is a good tool. A tool that takes longer than that to update is suboptimal.

Examples of tool pitfalls include the following:

- » Backlog items that encourage teams to write novels instead of having face-to-face conversations
- » Taskboard workflows that require the team to follow the tool's product development process rather than the process defined by the team
- » Inconvenient access to the tool
- » Reporting that encourages top-down or micromanaging behavior by stakeholders and managers
- » Lack of ease in updating or progressively elaborating requirements

From our experience, the most agile teams use 3x5 cards and sticky notes. Their taskboards adapt to the work at hand. Their backlogs are on walls with full transparency. Remote teams use digital tools that simulate a similar experience as physical tools. Often the teams we work with find that the cost to purchase certain tools and the complexity to train and administer them outweighs their benefits. Use the tool litmus test in Chapter 6 to better understand how to value individuals and interactions over processes and tools.

Before investing in expensive enterprise tools, consider first whether you've taken advantage of the lightweight approach to transparency and reporting of an agile approach, using scrum:

- » Access scrum teams' product backlogs and sprint backlogs, including burn-down charts on-demand. (See Chapters 9, 10, and 11 to learn more about product and sprint backlogs and burndown charts.)
- » Attend scrum teams' sprint reviews to see exactly what the teams have accomplished during the sprint.
- » Attend daily scrums — 15 minutes or less — to hear exactly what the team is working on that day and the impediments.
- » Engage with scrum teams about real-time truth and reality rather than waiting for delayed reports with limited context or opportunity to get instant clarification. Large organizations have successfully navigated agile waters with a simple spreadsheet. Refer to Chapter 11 for a simple release and sprint burndown spreadsheet.

When a sign of impeded agility due to tools arises, the following agile values and principles help navigate the situation:

- » Value #1: Individuals and interactions over processes and tools.
- » Principle #6: The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- » Principle #10: Simplicity — the art of maximizing the amount of work not done — is essential.

High 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. A high manager-to-creator ratio is another sign you're not agile.

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 and 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.

Empower people to self-organize in meeting the needs of your customers. You searched hard to find capable people through recruiting, training, and interviews. You hired them for their talent and experience. Leverage that investment by trusting them to get the job done. Minimize the investment you're making in people who don't create product.

Agile principles for navigating this situation follow:

- » Principle #5: Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- » Principle #11: The best architectures, requirements, and designs emerge from self-organizing teams.

Highly autonomous and highly aligned teams who are empowered to do what is best for the customer are more effective. Value autonomy, mastery, and purpose over traditional management command and control tendencies. See Chapter 8 to learn the benefits of people given autonomy, mastery, and purpose.

Working around What Scrum Exposes

Another sign you're not agile is if you work around the problems exposed by scrum's transparency rather than addressing them head on. Agile frameworks such as scrum are not problem-solving models; they are exposure models. The problems exposed probably always existed, but with scrum there is clear transparency and the problems stand out like a sore thumb. For the agile mindset to thrive, the problems — not just the symptoms, but the true root causes — must be addressed and removed. See Chapter 4 to learn more about root cause analysis techniques.

For example, Toyota car manufacturing used an andon cord on its manufacturing lines for years. An employee pulled the cord to stop the line immediately whenever a problem was found that needed to be corrected. Workers were not reprimanded for pulling the cord. Employees then worked together to fix the problem at the root so it was not passed further down the line, becoming more and more expensive to resolve. The same is true for addressing what scrum exposes: Address the problem at the root cause and prevent it from affecting anyone else in your organization, ever.

Watch out for the following signs:

- » Surface-level symptoms rather than the difficult root causes are addressed.
- » Scrum teams identify organizational constraints, keeping them from operating in more agile ways, but they are repeatedly ignored.
- » Scrum teams tend to skip sprint retrospectives because they feel that nothing ever changes.

These are clear signs that the leaders in the organization are not committed to the agile transformation change. If the elephant is in the room and everyone knows it, start eating the elephant one bite at a time.

Remembering the following agile principle will help:

- » Principle #12: At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Practicing Faux Agile

Last but not least is faux agile. Faux agile has the appearance of agile, but it's not.

In 2019, *Forbes'* Steve Denning shared the three laws of agile as a guide for organizations curious about the genuineness of their agile practices:

- » **The law of the customer:** An obsession with delivering value to customers as the be-all and end-all of the organization
- » **The law of the small team:** A presumption that all work be carried out by small, self-organizing teams, working in short cycles and focused on delivering value to customers
- » **The law of the network:** A continuing effort to obliterate bureaucracy and top-down hierarchy to operate as an interacting network of teams, all focused on working together to deliver increasing value to customers

Faux agile, according to Denning, demonstrates the following characteristics:

- » **Early-stage agile:** An agile journey never ends, so early on, the transition is incomplete. Results can be realized quickly from implementing a few agile practices or techniques, but time and experience are needed to stretch into more agile practices. The key is to avoid being fooled early on that “you’ve arrived.”
- » **Agile in name only:** For these organizations, you have to look beyond what they’re saying to how they’re operating. Do people demonstrate the agile mindset and move at the speed of a new startup? Or have they simply renamed existing non-agile practices with new agile terms?
- » **Agile for software only:** Although the agile movement started with software development, not only people creating software benefit from agile techniques. Don’t be deceived in thinking agile is only for software teams. For software teams to be successful, all parts of the organization — from senior leadership to human resources to finance to business development — must become agile.
- » **Stalled agile journeys:** When agile collides with unmoving traditional practices, the organization may be forced to practice faux agile. When the tension becomes severe, a leader may dismiss agile leaders and coaches by saying that the organization is already agile. Agile principles and bureaucracy are mutually incompatible: In the end, only one can survive.
- » **Branded agile:** From the Agile Manifesto and 12 principles have blossomed many different brands of agile. Beware of consultants and trainers who insist their branded version is the one and only true way.
- » **Scaling frameworks:** Scaling up agile is an anti-pattern. Progressive elaboration and decomposition to the most independent and valuable increments for the customer creates agility. Be selective regarding the various frameworks to use what works best for your organization.
- » **Agile lite:** Some organizations try to apply agile principles loosely without an agile mindset. Without an agile mindset, you’re left with an inert, lifeless set of ceremonies.

In addition to the preceding list, you may see other subtle, yet common signs of faux agile:

- » The organization continues to require even their scrum teams to follow a PMO-prescribed system development lifecycle (SDLC) with dated status reports and executive approval gates.

- » The organization tries a hybrid approach that combines agile techniques with traditional roles. This situation is like setting sail and dropping anchor at the same time and expecting to move. You have to make a choice. Faux agile organizations continue to support and recruit traditional roles in addition to or instead of the types of roles that enable agile principles. These roles are not the same and require very different skill sets. See Chapter 7 to learn more about scrum roles.
- » The organization uses technical product owners in addition to or in the absence of business- or customer-facing product owners.

If any of these ten anti-patterns exist in your organization, address them head on. The benefits of agile approaches to product development are extensive, proven, and worthwhile. Continually use agile values and principles to keep your journey to improved business agility on track.

IN THIS CHAPTER

- » Finding support for successful agile transitions
- » Getting involved with active agile communities
- » Accessing resources for common agile approaches

Chapter **24**

Ten Valuable Resources for Agile Professionals

Many organizations, websites, blogs, and companies exist to provide information about and support for agile product development. To help you get started, we compiled a list of ten resources that we think are valuable to support your journey to improved agility.

Agile Project Management For Dummies Online Cheat Sheet

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, and then type *Agile Project Management For Dummies* in the Search box.

Scrum For Dummies

In 2018, we published the second edition of *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.

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

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. To find a scrum user group in your area, search for your location at www.scrumalliance.org/resources/groups.

The Scrum Alliance site is also rich in blog entries, white papers, case studies, and other tools for learning and working with scrum. Chapter 18 lists many of the Scrum Alliance certifications.

The Agile Alliance

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, and presentations. Find an index of independent and local agile community groups across the world at www.agilealliance.org/communities/.

International Consortium for Agile (ICAgile)

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.

Mind the Product and ProductTank

www.mindtheproduct.com

Mind the Product is the world's largest community of people passionate about product. They also founded ProductTank meetups so that product leaders could connect, share, and learn from each other. With over 150,000 members worldwide, they offer blogs, global, regional and local events, local meetups, and training by leading product management experts from all over the world. The resources at ProductTank tend to be high quality, and the content is both unique and relevant to the issues facing agile product development teams. Find a local ProductTank meetup in your area at www.mindtheproduct.com/producttank.

Lean Enterprise Institute

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 product development, 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

<http://ronjeffries.com>

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>.

The Project Management Institute Agile Community

www.projectmanagement.com/practices/agile

The Project Management Institute (PMI) is the largest nonprofit project management membership association in the world. With nearly 3 million members in most countries throughout the world, 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 PMI-ACP certification along with access to papers, books, and seminars about agile project management.

Platinum Edge

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 20:

- » **Agile audits:** Auditing your current organizational structure and processes to create an agile implementation strategy that delivers bottom-line results. We also provide feedback on your current agile transition efforts to help you assess whether the investment you've made is generating the results you hoped for.
- » **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 teams, 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 certification classes:
 - Certified ScrumMaster (CSM)
 - Advanced Certified ScrumMaster (A-CSM)
 - Certified Scrum Product Owner (CSPO)
 - Certified Scrum Developer (CSD)
 - LeSS, Scrum@Scale, and SAFe approaches to scaling
 - PMI Agile Certified Practitioner (PMI-ACP) test preparation
- » **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.

Index

Numbers

80/20 rule, 85-86

A

Abbe, Ernest, 37

AC+OC>V formula, 243, 267, 305-306

Accredited Kanban Consultant (AKC) and Trainer (AKT), 370

Achor, Shawn, 156

ADKAR model, 399-400

Advanced Certified Scrum Product Owner (A-CSPO), 370

Advanced Certified ScrumMaster (A-CSM), 370, 453

affinity estimating, 195-196

Agile Alliance, 12, 22, 450-451

agile champion, 360, 363

agile coach. *See* agile mentors

agile concepts, 91-92, 239

Agile Manifesto. *See also* Agile Principles; agile project management

contracts in, 284

cross-industry application of, 24

discussion, 13, 22-23

documentation in, 27

focus of, 23

Four Values of

customer collaboration, 28, 278

flexibility, 29-30

overview, 24-25

people over process, 25, 219

working functionality, 26-28

history of, 21-22

litmus test, 47

agile mentors

discussion, 102, 134, 258

in pilot scrum team, 367-368, 371

product team and, 124

scrum team daily responsibilities of, 229

Agile Principles

customer satisfaction in, 32-34

discussion, 13, 31

grouping of, 31

history of, 22, 30. *See also* Platinum Principles

litmus test, 47

procurement and, 278-279

product development in, 38-42

quality in, 34-36

teamwork in, 36-38

agile procurement management

contracts in

closing, 286

cost structures, 282-283

creating, 283-285

discussion, 269, 278-279

needs determination in, 280-281

traditional versus, 279-280

vendors in

selecting, 281-283

working with, 281, 285-286

agile product development

adaptation in, 19

advantages of, 39-41

blank templates for

management comparison, 41

sprint backlog burndown chart,
221, 237

determining duration in, 290

discussion, 3, 16

inspections in, 19

litmus test, 47

minimizing scope bloat in, 19

principles of, 38-39

success rates of, 18

superiority of, 18-19

traditional product management versus

active products, 17-18

overview, 16-17

team permanence, 17

value maximization, 18

traditional project management
versus, 41

value cost formula in, 16

agile project management. *See also* risk management
adaptation in, 92
agile principles in
customer satisfaction, 32-34
overview, 31
quality, 34-36
attractive features of
to customers, 65-66
to development teams, 67-68
efficiency, 64-65
to executives, 64-65
greater value, 66-67
higher quality, 66
increased flexibility, 65-66
increased ROI, 65
less waste, 66-67
to managements, 66-67
benefits of
better metrics, 426-427
better product quality, 424
higher customer satisfaction, 423-424
higher team morale, 430
improved predictability, 429
increased collaboration, 426
increased ownership, 426
increased visibility, 427-428
iterative methodology, 50-52
more investment control, 428-429
optimized team structures, 429-430
overview, 49-50, 423
reduced risk, 425
blank templates for
customer satisfaction, 33
human interactions/processes, 26
identifying useful documentation, 27
certification, 104
changes caused by, 45-46
customer collaboration in, 28-29
customer satisfaction in, 53-54
discussion, 1, 7-8, 92-93
empirical control method, 14, 92
evolution of, 46
frameworks, 13
history of, 11-13
human interactions in, 25-26

inspection in, 92
litmus test, 47
methodologies, 13
mixing traditional and, 14
more information on, 2
Platinum Principles in
overview, 42
resisting formality, 42-43
teamwork, 43-44
visualization, 44-45
quality and, 34
superiority of, 23-24
synonyms in, 13-14
techniques, 13
tools, 13
traditional versus
failure detection, 63-64
flexibility, 55-57
overview, 54-55
project metrics, 62-63
quality, 60-61
ROI, 64
speed, 60-61
stability, 55-57
team performance, 61-62
unproductive activities, 57-60
transparency in, 92
waterfall project management versus
overview, 14-15
planning, 162
working functionality documentation in,
26-28
agile release train (ART) model, 390-391
Agile Retrospectives (Derby, Larsen), 249
agile scope management
agile principles related to, 270
change in
artifacts, 277-278
evaluating and prioritizing, 274-276
introducing, 274
overview, 272
understanding, 273-274
discussion, 269
time management and, 297-298
traditional versus, 270-272

agile transition. *See also* change management
committing to
 agile champion, 360
 individual, 359-360
 organizational, 358-359
 questions to consider, 361-362
 timing, 362-363
creating environment for
 overview, 368-369
 Roadmap to Value, 369
 training programs, 370-371
discussion, 357-358
key factors in successful
 addressing exposed issues, 433
 clear product vision, 433-434
 clear roadmap, 433-434
 coaching provided, 436
 collocation, 322, 432
 dedicated teams, 431-432
 definition of done, 433
 developer versatility, 434-435
 overview, 371, 431
 product owner empowerment, 434
 servant leadership, 435
 support for learning, 435-436
pilot scrum team in
 agile champion, 363
 agile mentor, 367-368, 371
 development team, 366
 overview, 364-365, 371
 product owner, 365-366
 scrum master, 366-367
 transition team, 364-365
systems thinking in, 364-365
warning signs in
 customer contact lacking, 440-441
 disengaged stakeholders, 439-440
 faux agile, 446-448
 ignoring exposed issues, 445-446
 lack of automation, 442
 limited skill versatility, 441
 long release cycles, 438-439
 manager-to-creator imbalance, 444-445
 non-shippable product increment, 437-438
 overview, 437
 prioritizing tools, 442-444

AKC (Accredited Kanban Consultant) and Trainer (AKT), 370
ART (agile release train) model, 390-391
artifacts
 for cost management, 299, 306
 discussion, 179
 in scope management, 277, 288
 in scrum approach, 102-103
 for time management, 298-299
automated testing, 232, 342-344

B

barely sufficient documentation, 27-28, 40, 181
Barton, Brent, 261
Basex, 136
Beck, Kent, 13, 105, 452
Beedle, Mike, 13
Bennekum, Arie van, 13
big bang development, 50-51
Blanchard, Kenneth, 315
Booch, Grady, 12
budgets, 300-303
bugs, 332
bulletin boards, 112, 115-116
burndown chart
 communicating progress with, 328-329
 discussion, 206
 sprint backlog, 220-222

C

Cagan, Marty, 71
CAL (Certified Agile Leadership), 370
capital expenditures (CapEx), 18, 256
capital redeployment, 305
cause-and-effect diagram, 87-88
CD (continuous deployment), 201, 231
Center for Applied Ethics, 315
Certified Agile Leadership (CAL), 370
Certified Enterprise Coach (CEC), 370
Certified Scrum Developer (CSD), 104, 370, 453
Certified Scrum Product Owner (CSPO), 104, 120, 370, 453
Certified Scrum Professional (CSP) for ScrumMasters (CSP-SM), Product Owners (CSP-PO), and Developers (CSP), 370

- Certified Scrum Trainer (CST), 104, 370
Certified ScrumMaster (CSM), 104, 120, 370, 453
Certified Team Coach (CTC), 370
change management. *See also agile transition discussion*, 395-396
executive ownership in overview, 412-413
servant leadership, 413-414
pitfalls to leadership pitfalls, 417-418
organizational pitfalls, 415-416
overview, 414
warning signs, 418-420
Platinum Edge model for overview, 401-402
Step 1: implementation, 403-404
Step 2: awareness, 404-405
Step 3: transition team, 405-407
Step 4: environment, 407-408
Step 5: training and recruiting, 408
Step 6: pilot and coaching, 408-410
Step 7: Roadmap to Value, 410
Step 8: gather feedback, 410-411
Step 9: solidify improvements, 411
Step 10: progressively expand, 412
resistance in, 396-397
SCARF model in, 396
tools for ADKAR model, 399-400
Kotter model, 400-401
Lewin model, 398
overview, 397
cheat sheet, 2, 449
chief product owner (CPO), 387
Christensen, Clayton, 75-77
CI (continuous integration), 35, 201, 231, 339
Cirillo, Francesco, 136
Cockburn, Alistair, 13, 112, 325
collaboration tools, 118, 120-121, 323
collective code ownership, 339
collocation key for transition, 432
success rate of team, 322
team, 61, 110-112
communication agile principles in, 324-325
agile versus traditional, 324
among dislocated teams, 111-112, 322-323
discussion, 307
effectiveness of different forms of, 112
face-to-face effectiveness, 325-326
goals, 114
overview, 111-112, 114, 117
tools, 115-116
fidelity, 112
human interactions and, 26
inefficiency of email for, 58-59
methods for, 325-328
openness and, 137-138
osmotic, 111-112
product owner and, 125-126
progress tracking and, 328-330
resisting formality in, 42-43
with stakeholders, 39, 125-126
team size limits and, 146
technological tools for collaboration tools, 118, 323
constraints, 121
overview, 116
persistent chat, 117
videoconferencing, 117, 322
virtual collaboration board, 120-121
in waterfall project management, 53
in XP approach, 105
communication fidelity, 112
community of practice (CoP), 158, 383-384
comprehensive documentation, 26-28
context switching. *See multitasking*
continuous deployment (CD), 201, 231
continuous integration (CI), 35, 201, 231, 339
contract negotiation, 28-29
Conway, Melvin, 394
Conway's Law, 394
CoP (community of practice), 158, 383-384
cost management agile principles in, 299
agile versus traditional, 300

discussion, 287, 299
lowering costs in, 304-306
using artifacts for, 299, 306
velocity and, 303-306
COVID-19 pandemic, 109-110, 119, 321
CPO (chief product owner), 387
credentials. *See* specific certifications
Crossing the Chasm (Moore), 167
Crystal methodology, 13
CSD (Certified Scrum Developer), 104, 370, 453
CSM (Certified ScrumMaster), 104, 120, 370, 453
CSP (Certified Scrum Professional) for ScrumMasters (CSP-SM), Product Owners (CSP-PO), and Developers (CSP), 370
CSPO (Certified Scrum Product Owner), 104, 120, 370, 453
CST (Certified Scrum Trainer), 104, 370
CTC (Certified Team Coach), 370
Cunningham, Ward, 13, 105, 452
customer representative. *See* product owner
customers
 attraction to agility, 65-66
 collaboration with, 28-29, 278
 demands of, 70
 determining needs of
 customer-focused goals, 82
 embracing early failure, 81
 liberating structures, 83-84
 overview, 79
 root cause analysis (RCA), 84-88
 scientific method, 79-80
 story maps, 83
discussion, 69-70
identifying
 empathy maps, 74-75
 interviews, 78
 job-to-be-done approach, 75-78
 journey maps, 74-75
 overview, 71
 product canvas and, 71-74
 product discovery workshops and, 79
involvement of, 53-54
satisfaction of, 32-34, 423-424
stakeholders as, 36
uncertainty and, 70

D

DA (Disciplined Agile) toolkit, 392-394
daily scrum. *See also* Roadmap to Value; shippable functionality; sprints
 addressing roadblocks in, 217, 234-235, 354
 discussion, 103, 114, 215-216
 duration of, 217
 guidelines for, 217-218
 purpose of, 218
 scrum master and, 58, 156
 scrum teams and, 164
 standing during, 217-218
 topics addressed in, 216-217
death march, 40
decomposition of requirements. *See* requirements
definition of done
 discussion, 26-27, 36
 estimation poker and, 194
 product increment and, 240-241
 risk reduction in, 348-349
Demarco, Tom, 265
demonstrations, 59
dependencies, 178, 199
Derby, Esther, 249
development operations (DevOps), 201
development teams. *See also* multiple teams; team dynamics
 building capability in, 157-158
 building knowledge in, 157-158
 crucial role of, 54
 discussion, 35, 124
 human interactions in, 26
 long-term
 advantages, 149-150
 characteristics of effective, 153
 knowledge, 150-151
 overview, 149
 Shu Ha Ri concepts, 153-154
 team development phases, 151-153
 working agreement, 152, 154-155
 members of, 128-130
 motivating
 alignment, 157
 autonomy, 155, 157
 mastery, 156

development teams (*continued*)

overview, 155
purpose, 156
philosophy of
 cross-functionality, 141-142
 dedicated team, 140-141
 overview, 139-140
 ownership, 147
 self-management, 144-145
 self-organization, 143-144
 size limits, 146-147
product development and, 230-231
project management and, 21-22
quality management by, 334
in scrum approach, 101-102
scrum team daily responsibilities of, 226
sprint backlog and, 102, 224
DevOps (development operations), 201
Digital.ai, 100
Disciplined Agile (DA) toolkit, 392-394
dislocated teams, 111-112, 321-323. *See also* remote work; team dynamics
distractions
 common, 113
 Pomodoro technique for avoiding, 136
scrum master and, 61-62, 113, 231
velocity and, 295
documentation, 27-28, 59-60
double work agile, 329
Drive (Pink), 155
Drucker, Peter, 82
DSDM approach, 12

E

earned value management (EVM), 329
EAT (executive action team), 386
Economics of Fatigue and Unrest, The (Florence), 37
Economy, Peter, 83
Edmondson, Amy, 413
email, 58-59
empathy maps, 74-75
empirical control method, 14, 17, 80
EMS (executive metascrum), 387-388
end user. *See* customers
epic user stories, 173, 190

Essentialism (McKeown), 262

estimation poker, 192-194
EVM (earned value management), 329
executive action team (EAT), 386
executive metascrum (EMS), 387-388
Extreme Programming Explained (Beck), 105
extreme programming (XP)
 continuous deployment (CD) in, 201
 continuous integration (CI) in, 35, 201
 discussion, 40, 92, 105, 452
 key practices of, 106-107, 231
 other approaches and, 107-108
 principles of, 105-106
 quality management in, 338

F

facilitator. *See* scrum master
FDD approach, 12
feature requirements, 173
feedback, 50, 70
Fibonacci sequence, 190, 193
Fifth Discipline, The (Senge), 150
fishbone diagram, 87-88
fist of five, 133
“5 Simple Rules of Agile Portfolio Management, The” (Barton), 261
five why’s, 87
flexibility, 29-30
Florence, P. Sargent, 37
formality, 43, 50, 114-115
Fowler, Martin, 13

G

gold-plating documentation, 27-28
Greanleaf, Robert K., 315
Greenleaf Center for Servant Leadership, 315
Grenning, James, 13
Gudith, Daniela, 265

H

hardware, 8
Harvard Business Review, 12-13, 156
Harvard Business School, 75

Hightsmith, Jim, 13
human interactions, 25-26
Hunt, Andrew, 13

I

ICAgile (International Consortium for Agile), 370, 451
icons, 2
IEEE (Institute of Electrical and Electronics Engineers), 8
IID techniques, 12
Implementing Lean Software Development (Poppendieck, Poppendieck), 151
information radiator, 115, 198, 235-236, 330
information technology (IT) industry, 21, 201
Inspired (Cagan), 71
Institute of Electrical and Electronics Engineers (IEEE), 8
International Consortium for Agile (ICAgile), 370, 451
INVEST approach, 192
IRR (internal rate of return), 255-256
Ishikawa, Kaoru, 87-88
Ishikawa diagram, 87-88
IT (information technology) industry, 21, 201
iterations. *See also* sprints
 discussion, 14, 32, 51
 history of, 95
 short duration of, 35, 40, 56
 in waterfall project management, 94-95
 Winston Royce on, 94-95
iterative methodology, 12, 50-52

J

Jeffries, Ron, 13, 105, 452
job-to-be-done approach, 75-77
Johnson, Spencer, 315
Jones, Daniel T., 96
journey maps, 74-75
just-in-time (JIT) approach, 96, 161, 164-165. *See also* product roadmap; Roadmap to Value

K

kanban
 board, 115, 117
 discussion, 97-99
 operational support and, 202
 task board and, 223

Kanban Coaching Professional (KCP), 370
Kanban Management Professional (KMP I, II), 370
Kanban University, 370
Kellerman, Gabriella Rosen, 156
Kennedy, John F., 82
Kern, Jon, 13
Ketterin, Charles, 79
Kilburn, Tom, 93
KMP I, II (Kanban Management Professional), 370
Kotter, John, 400

L

large-scale scrum (LeSS), 380-384
Larsen, Diana, 249
law of diminishing returns, 267
Layton, Mark, 11, 21
lean canvas, 72
lean coffee approach, 158
Lean Enterprise Institute, 451
lean product development
 discussion, 92, 95
 JIT elaboration in, 96
 kanban practices in, 97-99
 other approaches and, 107-108
 principles of, 96-97
Lean Software Development (Poppendieck, Poppendieck), 96
Lencioni, Peter, 72
LeSS (large-scale scrum), 380-384
Lewin, Kurt, 398
Lewin model, 398
liberating structures, 83-84, 86
linear methodology, 50-52
Lipmanowicz, Henri, 83

M

Machine that Changed the World, The (Womack, Jones, Roos), 96
macrostructures, 84
Management by Objectives, 82
"Managing the Development of Large Software Systems" (Royce), 8, 12, 93
Manifesto for Agile Software Development, 13. *See also* Agile Manifesto
Marick, Brian, 13

Mark, Gloria, 265
Mark II Aiken Relay Calculator, 332
marshmallow challenge, 50-51
Martin, Robert C., 13
mass-production methods, 95-96
Maxwell, John C., 81
McCandless, Keith, 83
McKeown, Greg, 262
Mehrabian, Albert, 112
Mellor, Steve, 13
microstructures, 84
mind maps, 87
Mind the Product, 451
minimal marketable features, 197-198, 289
minimum viable product (MVP), 81, 83, 438
Moltke, Helmuth von, 162
Moore, Geoffrey, 167-168
multiple teams
 common challenges with, 374-375
 DA toolkit for, 392-394
 discussion, 373-374
 LeSS for
 combined meetings, 383-384
 combined sprint review, 382-383
 communities of practice (CoP),
 383-384
 daily scrum, 383
 huge framework, 381-382
 overview, 380
 smaller framework, 380-381
 SAFe for
 configurations, 388-389
 core competencies, 390-391
 joint program PI, 391-392
 managers, 392
 Scrum@Scale approach for
 overview, 384-385
 product owner cycle, 387-388
 scrum master cycle, 385-386
 synchronization, 388
vertical slicing for
 overview, 376
 scrum of scrums model, 376-379
multitasking, 113, 140, 153, 265-266
MVP (minimum viable product), 81, 83, 438

N

"New New Product Development Game, The" (Takeuchi,
Nonaka), 12-13, 153
New One-Minute Manager, The (Blanchard, Johnson), 315
nine why's, 87
Nonaka, Ikujiro, 13, 153
Nordstrom Innovation Lab, 81

O

Ohno, Taiichi, 247
One Minute Manager (Blanchard, Johnson), 315
one-day sprints, 202
operational expenditures (OpEx), 256
osmotic communication, 111-112

P

pair programming, 43, 106, 232, 338
PAL I (Professional Agile Leadership), 370
Pareto, Vilfredo, 85
Pareto rule, 85-86
Patton, Jeff, 83
PDSA (Plan-Do-Study-Act) approach, 11-12
peer review, 232, 338-339
pen-pencil rule, 200
permanent teams. *See* development teams
personas, 187-189, 205
physical environments
 communication and
 face-to-face, 114-117
 information radiator, 115
 technological tools, 116-117
 creating
 dedicated areas, 112
 removing distractions, 113
 team collocation, 110-112
 discussion, 109-110
 tools for
 constraints, 121
 litmus test, 119
 overview, 118
 purpose, 119
 success-encouraging tools, 119-121
 virtual collaboration board, 120-121

- PI (program increment) planning, 391-392
- Pink, Daniel H., 155
- Plan-Do-Study-Act (PDSA) approach, 11-12
- planning. *See also* product roadmap; Roadmap to Value
- adaptation stage of, 165
 - decomposition of requirements in, 164
 - discussion, 161
 - excessive, 50
 - inspection stage of, 165
 - JIT elaboration in, 161-162
- planning poker, 192-194
- Platinum Edge
- discussion, 21
 - forced team dislocation and, 120
 - history of, 12
 - services offered by, 368, 452-453
- Platinum Principles, 30, 42-45
- PMI (Project Management Institute), 104, 452
- PMI Agile Certified Practitioner (PMI-ACP), 104, 452
- PMI-ACP (PMI Agile Certified Practitioner), 104, 452
- PMI-ACP (Project Management Institute Agile Certified Practitioner) accreditation, 371
- pods, 112
- Pomodoro technique, 136
- Poppendieck, Mary, 96
- Poppendieck, Tom, 96
- portfolio management
- agile principles in, 253-255
 - CapEx in, 256
 - discussion, 253-254
 - forecasting investment returns in
 - agile mentors, 258
 - overview, 256
 - product mix, 260-261
 - risk-value matrix, 257-258
 - short- versus long-term decisions, 259-260
 - value and risk prioritization, 257-259
 - IRR in, 255-256
 - managing
 - avoiding multitasking, 265-266
 - continuous prioritization, 267-268
 - engaging vendors, 264
 - incremental funding, 264
 - keys to effectiveness, 264-265
 - other factors, 262, 264
 - overview, 261
- reducing complexity, 261-262
- reducing investments, 266-267
- revising investments, 268
- visualization, 262-263
- multitasking and, 265-266
- OpEx in, 256
- other costs to consider in, 256
- SWOT analysis in, 254
- Practice of Management, The* (Drucker), 82
- processes, 25-26
- product backlog. *See also* release planning
- communicating progress with, 328-329
 - completing, 180-182
 - continually updating, 189
 - cost management and, 299
 - discussion, 27, 175, 179
 - estimates, 182
 - Pareto rule with, 86
 - product owner and, 224
 - product roadmap and, 180
 - risk management and, 353
 - in scrum approach, 102
 - story maps and, 83
 - time management and, 299
 - user stories and, 182
- product canvas, 71-74
- product discovery, 79, 127-128
- product increment, 102-103, 240-241. *See also* shippable functionality; sprint review
- product mix, 260-261
- product owner
- characteristics of good, 125-127
 - communication and, 114, 125-126
 - daily scrum and, 217
 - discussion, 32-33, 53-54
 - product backlog and, 102, 224
 - product development and, 128, 230
 - product discovery and, 127-128
 - quality management and, 334, 339-340
 - responsibilities of, 124-125
 - review of user stories, 233
 - in scrum approach, 101
 - scrum team daily responsibilities of, 225, 236-237
 - sprint review meeting and, 245
 - team support from, 61

- product roadmap. *See also* product backlog; user stories
cost management and, 299
creating
arranging product features, 175-176
assessing value and risk, 178-179
defining product requirements, 173-175
determining time frames, 180
estimating and prioritizing, 176-180
identifying stakeholders, 172-173
overview, 171-172
saving, 180
updating, 180
discussion, 162
product canvas and, 74
risk management and, 353
in scrum approach, 108
time management and, 299
product scope, 270
product team, 124
product vision statement
creating
drafting, 167-169
finalizing, 170
overview, 165-166
product objective, 167
revising, 169-170
validating, 169-170
discussion, 162, 340, 353
product canvas and, 74
in scrum approach, 108
ProductTank, 451
Professional Agile Leadership (PAL I), 370
Professional Scrum Developer (PSD I), 370
Professional Scrum Master (PSM I, II, III), 370
Professional Scrum Product Owner (PSPO I, II, III), 370
program increment (PI) planning, 391-392
progressive elaboration of requirements. *See*
requirements
project management, 8-12. *See also* agile project
management; traditional project management
Project Management Institute Agile Certified Practitioner
(PMI-ACP) accreditation, 371
Project Management Institute (PMI), 104, 452
Project Mercury, 12
project scope, 270
Prosci ADKAR model, 399-400
- PSD I (Professional Scrum Developer), 370
PSM I, II, III (Professional Scrum Master), 370
PSPO I, II, III (Professional Scrum Product Owner), 370
- Q**
- quality management
agile principles in, 331-332
in agile project management, 424
daily testing and, 335
by development teams, 334
discussion, 34-36, 331
feedback loops in, 333
proactive
acceptance criteria, 340
collective code ownership, 339
continuous integration (CI), 339
development techniques, 338
face-to-face communication, 340-341
good design, 337-338
organizational commitment, 337
overview, 335-336
pair programming, 338
peer review, 338-339
role of product owner, 339-340
sustainable work pace, 341
technical excellence, 337-338
risk and, 333
traditional versus agile, 332-333
via automated testing, 342-344
via inspecting and adapting, 341-342
- R**
- R&D (research and development), 80
“Racing in Reverse,” 40
RCA (root cause analysis), 84-88
Reece, Andrew, 156
refactoring, 105-106
relative estimating, 13, 108, 178
release planning. *See also* sprints
cost management and, 299
creating, 198-200
discussion, 82, 163
key activities of, 197-198
minimal marketable features, 198

- minimal marketable features and, 197
- product deployment in
- market preparation, 204-205
 - operational support, 201-203
 - organizational activities, 203-204
- overview, 200-201
- risk management and, 354
- in scrum approach, 108
- sprints and, 199-200
- time management and, 299
- in XP approach, 106
- release train engineer (RTE), 390
- remote work. *See also* dislocated teams; team dynamics
- communication and
 - overview, 111-112, 116
 - tools, 117-121
 - discussion, 109-110
- requirements, decomposition of
- affinity estimating, 195-196
 - epic user stories, 190
 - estimation poker, 193-194
 - features, 173, 190
 - guide, 190-191
 - INVEST approach, 192
 - overview, 164, 183
 - during sprints, 190
 - tasks, 174
 - themes, 173, 190
 - user stories, 174
- research and development (R&D), 80
- return on investment (ROI), 167
- Ries, Eric, 81
- risk management
- agile principles in, 345
 - in agile project management, 54, 425
 - agile versus traditional, 345-347
 - artifacts and meetings for, 353-354
 - end-of-sprint and, 241
 - inherent risk reduction in
 - definition of done, 348-349
 - failing fast concept, 351-352
 - overview, 348
 - self-funding development, 349-351
 - quality and, 333
 - in waterfall project management, 53
- risk-value matrix, 257-258
- roadblocks, 207, 217, 234-235
- Roadmap to Value. *See also* product backlog; release planning
- adaptation stage in, 165
 - creating, 180
 - decomposition of requirements in, 164
 - discussion, 369
 - inspection stage in, 165
 - product roadmap in
 - arranging product features, 175-176
 - assessing value and risk, 178-179
 - creating, 171-172
 - defining product requirements, 173-175
 - estimating and prioritizing, 176-180
 - identifying stakeholders, 172-173
 - saving, 180
 - updating, 180
 - product vision statement in
 - creating, 166-170
 - overview, 165-166
 - release planning in
 - creating, 198-200
 - key activities, 197-198
 - minimal marketable features, 197-198
 - release sprints, 199-200
 - scope management and, 273-274
 - stages of, 162-164
- Robichaux, Alexi, 156
- Rock, David, 396
- ROI (return on investment), 167
- role
- of agile mentor, 134
 - of development team members, 128-130
 - discussion, 123-124
 - of product owner
 - overview, 124
 - product development, 128
 - product discovery, 127-128
 - responsibilities, 124-126
 - of scrum master
 - characteristics, 131-132
 - responsibilities, 130-131
 - of stakeholders, 125, 132-134

Roos, Daniel, 96
root cause analysis (RCA), 84-88
Royce, Winston, 8-9, 12, 93
RTE (release train engineer), 390

S

Scaled Agile Framework (SAFe), 388-392
SCARF model, 396
Schwaber, Ken, 12-13
scientific method, 79
scope bloat, 10
scope creep, 272
Scrum Alliance, 12, 24, 104, 370, 450
scrum approach. *See also* daily scrum
 additional features of, 108
 agile mentors in, 102
 artifacts in, 102-103
 customer satisfaction and, 32
 discussion, 92, 100
 events in, 103
 history of, 12-13
 other approaches and, 107-108
 roles in, 101-102
 sprints in, 100-101
 stakeholders in, 102
 terminology in, 53
Scrum For Dummies (Layton), 11, 450
scrum master
 characteristics of good, 131-132, 144-145
 coaching provided by, 230
 discussion, 53-55, 102
 distractions prevented by, 61-62, 113, 231
 meetings and, 58, 156
 quality management and, 334
 responsibilities of, 130-131
 roadblocks addressed by, 207, 217, 234
 scrum team daily responsibilities of, 227, 236-237
 servant-leader role of, 314-315
 sprint retrospective meetings and, 249-250
scrum room, 112
scrum teams. *See also* daily scrum; multiple teams;
 physical environments; team dynamics
 communication among dislocated, 111-112
 daily meetings, 164
 discussion, 54-55, 92, 123-124

philosophy of
 cross-functionality, 141-142
 dedicated team, 140-141
 overview, 139-140
 ownership, 147
 self-management, 144-145
 self-organization, 143-144
 size limits, 146-147
pilot
 agile mentor, 367-368
 development team, 366
 overview, 364-365
 product owner, 365-366
 scrum master, 366-367
 stakeholders, 367
product discovery workshops and, 79
sprint daily responsibilities of
 agile mentors, 228-229
 development team members, 226
 overview, 224-225, 236-237
 product owner, 225
 scrum master, 227
 stakeholders, 228
sprint reviews and, 103
stakeholders and, 102
Scrum@Scale approach, 384-388
self-management
 excellence product of, 337
 of scrum master, 144-145
 team dynamics and, 309-313
Senge, Peter, 150
servant leadership, 314, 413-414, 435
shadowing, 43
shared drives, 109-110
Sherwart, Walter, 11-12
shippable functionality. *See also* user stories
 developing in, 230-231
 discussion, 229-230
 elaborating in, 230
 sprint review and, 240-241
 verifying in
 automated testing, 232
 identifying roadblocks, 234-235
 mob programming, 232-233
 overview, 231

- pair programming, 232
peer review, 232
product owner review, 233
Shu Ha Ri concepts, 153-154
Sinek, Simon, 166
Slack, 117
Slack (Demarco), 265
SMART goals, 82
software development, 8, 93-94, 201
Software Development as a Cooperative Game (Cockburn), 325
Spears, Larry, 314
Spira, Jonathan, 136
sprint backlog
 communicating progress with, 328-329
 cost management and, 299
 creation of, 206-207
 development teams and, 102, 224
 discussion, 27
 overview, 206
 risk management and, 354
 in scrum approach, 102
 time management and, 299
 tracking progress with
 burndown chart, 206, 220-222
 end of day, 237-238
 overview, 219
sprint planning
 components of, 206
 discussion, 82, 205
 duration of, 205
 meeting for
 create backlog tasks, 211-212
 overview, 207-209
 risk management, 354
 selecting user stories, 209-210
 setting goals, 209-210
 task guidelines, 212-213
sprint retrospective. *See also* Roadmap to Value
 adapting in, 250
 discussion, 103, 164, 239, 245
 goal of, 246-247
 inspecting in, 250, 342
 meeting
 duration, 248
 guidelines, 249
 scrum master, 249-250
 topics for inspection, 248-249, 354
 planning for, 247
sprint review. *See also* Roadmap to Value
 communicating progress with, 328-329
 determining future investment during, 243
 discussion, 84, 164, 239-240
 meeting
 collecting feedback, 244-245, 342
 duration, 242
 feedback cycle, 241-242, 354
 guidelines, 242-244
 overview, 241
 product owner tasks, 245
 scheduling, 242
 preparation for, 240
 scrum teams and, 103
 shippable functionality and, 240-241
 value of, 244
sprints. *See also* release planning; scrum teams; user stories
 discussion, 14, 32, 51, 53
 planning for, 103, 163
 release planning and, 199-200
 requirement decomposition during, 190-191
 scientific method and, 80
 in scrum approach
 adaptations, 101
 inspections, 100-101
 overview, 100
 scrum team daily responsibilities and
 agile mentors, 229
 development team members, 226
 end of day, 236-237
 overview, 224-225
 product owner, 225
 scrum master, 227
 stakeholders, 228
 shippable functionality and
 developing, 230-231
 elaborating, 230
 identifying roadblocks, 234-235
 overview, 229-230
 verifying, 231-233
 short duration of, 35, 40, 56
 stakeholders and, 103
 testing during daily, 335

- stakeholders
- communication with, 39, 125-126
 - creating user stories with, 187-189
 - as customer, 32
 - discussion, 132-134
 - disengaged, 439-440
 - pilot scrum teams and, 367
 - product discovery workshops and, 79
 - product roadmap and, 172-173
 - product team and, 124
 - role of, 125
 - in scrum approach, 102
 - scrum team daily responsibilities of, 228
 - scrum teams and, 102
 - sprints and, 367
 - user stories and, 186-187
- Standish Group
- "2015 Chaos Report," 345-346
 - study on project success/failure rates, 11, 18
 - study on scope bloat, 10
- stay-at-home orders, 109-110, 119
- story maps, 83
- Sutherland, Jeff, 12-13
- swarming, 108, 141-142, 211
- SWOTs (strengths, weaknesses, opportunities, and threats), 254
- system architect/engineer, 390-391
- systems thinking, 364-365
- T**
- Takeuchi, Hirotaka, 13, 153
- task board
- communicating progress with, 328-330
 - kanban similar to, 223
 - risk management and, 354
 - tracking progress with
 - components, 222-223
 - overview, 219, 222
 - reading, 223-224
- task requirements, 174
- TDD (test-driven development), 106, 338
- team coach. *See scrum master*
- team dynamics. *See also multiple teams*
- agile principles in, 307-308
 - agile versus traditional, 308-309, 429-430
- collocation and, 61
- discussion, 307
- managing
- cross-functionality, 317-319
 - dedicated teams, 316-317
 - dislocated teams, 321-323
 - limited size, 320-321
 - openness, 319-320
 - overview, 309
 - self-management, 309-313
 - servant leadership, 314-315
- Team Kanban Practitioner (TKP), 370
- team room, 112
- Teams, 117
- teams. *See also team dynamics*
- dedicated, 140-141, 431-432
 - discussion, 26, 36-37
 - focus, 61-62
 - in Platinum Principles, 43-44
 - stakeholders and, 36
 - strategies for effective, 38
 - support for, 61
 - tighter control and, 62-63
 - visualization techniques and, 71-72
 - in waterfall project management, 53
- technical excellence, 35, 337-338
- telepresence robots, 117
- test-driven development (TDD), 106, 338
- testing
- during daily sprints, 335
 - discussion, 35-36, 51, 60
 - in XP approach, 105
- theme requirements, 173, 190
- Thomas, Dave, 13
- thrashing. *See multitasking*
- time management. *See also velocity*
- advantages of agile, 288
 - agile principles and, 287-288
 - agile versus traditional, 288
 - discussion, 287
 - lowered costs and, 305
 - multiple teams for, 298
 - scheduling in, 289
 - scope changes and, 297-298
 - using artifacts for, 298-299

TKP (Team Kanban Practitioner), 370
Toyota Production System
 discussion, 247
 just-in-time (JIT) elaboration and, 96
 kanban practices and, 97, 223
traditional procurement management, 279-280
traditional project management
 agile product development versus, 41
 agile versus
 failure detection, 63-64
 flexibility, 55-57
 overview, 54-55
 project metrics, 62-63
 quality, 60-61
 ROI, 64
 speed, 60-61
 stability, 55-57
 team performance, 61-62
 unproductive activities, 57-60
cost management in, 299-300
customer involvement in, 28-29
death march in, 40
inflexibility of, 23, 30
modern products and, 50
planning in, 161
risk management in, 53
scope in, 270
unrealistic nature of, 50-51
traditional scope management, 270-272
Tuckman, Bruce, 151
Tuckman phases to performance, 151-153

U

uniform product developer titles, 44
useful documentation, 27-28
user stories. *See also* shippable functionality; sprints
 creating
 acceptance criteria of, 184
 advantages, 186
 determining product requirements, 189-190
 electronic tools, 185
 identifying stakeholders, 186-187
 identifying users, 187-188
 three C's formula, 185

decomposition of requirements in, 174
discussion, 13, 82, 183, 290
INVEST approach and, 192
product backlog and, 182
product owner review of, 233
requirement decomposition in
 affinity estimating, 195-196
 estimation poker, 192-194
 in scrum approach, 108
User Story Mapping (Patton, Economy), 83

V

value cost formula, 16
values
 discussion, 134-135
 establishing new
 commitment, 135-136
 courage, 138-139
 focus, 136-137
 openness, 137-138
 respect, 138
velocity
 adjusting, 291
 attaining consistent, 296-297
 calculating, 291-292
 cost management and, 303-304
 for determining other costs, 306
 discussion, 199
 estimating timeline using, 292-294
 increasing, 246, 294-295, 304
 for long-range planning, 289-290
 monitoring, 291
 in scrum approach, 108
vertical slicing, 376-379
videoconferencing tools, 117, 120
virtual collaboration board, 120-121
visualization activities
 for determining customer needs, 83
 discussion, 44-45
 for identifying customers
 empathy maps, 74-75
 journey maps, 74-75
 product canvas for, 71-72, 74

W

Wake, Bill, 192
waterfall project management
 agile versus, 14-15, 162, 346
 discussion, 8-9, 12, 14
 failure in agile versus, 351-352
 inflexibility of, 22, 30
 iterations in, 94-95
 major aspects of, 53
 origins of, 93-94
 risk management in, 53
 scope creep in, 283
 unrealistic nature of, 50-51

whiteboards, 111-112, 115-116

Womack, Japes P., 96

working agreement, 154-155

working functionality

 comprehensive documentation versus,
 26-28

 discussion, 328

 product increment and, 102-103

Wujec, Tom, 50

X

XP. *See* extreme programming

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 and program management field. He is the 2020 president of the Project Management Institute (PMI) Southern Nevada Chapter as well as the Los Angeles chair for the Agile Leadership Network. Mark is the author of the international *Scrum For Dummies* and *Agile Project Management For Dummies* book series, creator of the Agile Foundations Complete Video Course, and is the founder and managing member of Platinum Edge, LLC — an enterprise transformation company that uses organizational design to help businesses with their agile transformation journey.

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, DevSecOps, and other agile solutions.

Additional information can be found at <https://platinumedge.com>.

Steven J Ostermiller is a trainer, coach, and mentor helping organizations evolve to maximize business value and minimize risk through lean and agile principles and practices. He is the founder and executive director of Utah Agile (in partnership with Agile Alliance, Scrum Alliance, and Silicon Slopes), a non-profit professional community committed to increasing agility for Utah businesses, technology, and individuals. Steve developed and taught the agile project management curriculum for Ensign College in Salt Lake City, Utah and serves on its project management advisory board. He was technical editor of *Scrum For Dummies* and Pearson Education's Agile Foundations Complete Video Course.

Steve's expertise comes from nearly 20 years of successes and failures as a project manager, product manager, operations executive, scrum master, and agile coach,

trainer, and consultant. He has worked with executive leadership and product development teams in a variety of industries on the Fortune lists. He is a Scrum Alliance Certified Scrum Trainer (CST), an ICAgile Certified Professional in Coaching and Facilitation (ICP-ACC, ICP-ATF), and a Project Management Professional, and holds a B.S. in Business Management/Organizational Behavior from the Marriott School of Management at Brigham Young University.

Dean J. Kynaston is an experienced scrum master, coach, and mentor with nearly 20 years of experience empowering leaders, teams, and individuals to become more agile. With Steve, he taught the agile project management curriculum at Ensign College in Salt Lake City, Utah. Dean was also a technical editor for *Scrum For Dummies* and the author of multiple Platinum Edge blog articles.

A Platinum Edge alumni himself and former Project Management Professional (PMP), Dean has worked with multiple organizations and seen much success applying agile values and principles. He has worked with executive leadership and individual teams in both for-profit and non-profit industries, particularly in real estate, construction, automotive, healthcare, and pharmaceuticals. He holds an MBA from Boise State University, is a Certified Scrum Professional (CSP-SM and CSP-PO), and earned a B.S. in Business Management with an emphasis in finance from the Marriott School of Management at Brigham Young University. As a busy father of eight children, Dean finds many opportunities to use scrum with his family team as well.

Dedications

To my special snowflake. I'm so thankful to have you in my life and to be sharing this wild ride with you. The world hasn't seen anything yet. — Mark

To Gwen, my complete and final answer. And to our five littles, who give me every reason to continuously inspect and adapt. — Steve

To my angel mother, an amazing woman and nurturer who led by example in "enduring to the end." — Dean

Authors' Acknowledgments

We'd like to again thank the numerous people who contributed to the previous editions of this book and helped make it a reality. We're also very grateful to those who helped make this third edition a more valuable field guide: Andrew Workmon, for his insight and technical editing; Caroline Patchen for yet again ensuring that these concepts are more easily understood through clear visualization; Craig Larman, Bas Vodde, Jeff Sutherland, and Dean Leffingwell for providing scaling options to the public and for their valuable feedback on the de-scaling chapter; to Susan Pink and Debbye Butler; and to Steve Hayes 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

Executive Editor: Steve Hayes

Project Editor: Susan Pink

Copy Editor: Susan Pink

Technical Editor: Andrew Workmon

Proofreader: Debbye Butler

Sr. Editorial Assistant: Cherie Case

Production Editor: Mohammed Zafar Ali

Cover Image: © 3d_kot/Shutterstock