O'REILLY®

Learning Agile

UNDERSTANDING SCRUM, XP, LEAN, AND KANBAN

Andrew Stellman & Jennifer Greene

O'REILLY®

Learning Agile

Agile has revolutionized the way teams approach software development, but with dozens of agile methodologies to choose from, the decision to "go agile" can be tricky. This practical book helps you sort it out, first by grounding you in agile's underlying principles, then by describing four specific—and well-used—agile methods: Scrum, extreme programming (XP), Lean, and Kanban.

Each method focuses on a different area of development, but they all aim to change your team's mindset—from individuals who simply follow a plan to a cohesive group that makes decisions together. Whether you're considering agile for the first time, or trying it again, you'll learn how to choose a method that best fits your team and your company.

- Understand the purpose behind agile's core values and principles
- Learn Scrum's emphasis on project management, self-organization, and collective commitment
- Focus on software design and architecture with XP practices such as test-first and pair programming
- Use Lean thinking to empower your team, eliminate waste, and deliver software fast
- Learn how Kanban's practices help you deliver great software by managing flow
- Adopt agile practices and principles with an agile coach

"What Andrew and Jenny have done is create an approachable, relatable, understandable compendium of what agile is. You don't have to decide in advance what your agile approach is. You can read about all of them, and then decide. On your way, you can learn the system of agile and how it works."

—Johanna Rothman
Author and Consultant,
www.jrothman.com

Andrew Stellman and **Jennifer Greene** have been building software and writing about software engineering since 1998. They founded Stellman & Greene Consulting in 2003, and continue to work with software teams every day to build and deliver software to their users. Other O'Reilly titles they've written include *Beautiful Teams*, *Head First C#*, *Head First PMP*, and *Applied Software Project Management*.

SOFTWARE DEVELOPMENT/AGILE

US $44.99 **CAN $47.99**

ISBN: 978-1-449-33192-4

Twitter: @oreillymedia
facebook.com/oreilly

Praise for *Learning Agile*

Another amazing book by the team of Andrew and Jennifer. Their writing style is engaging, their mastery of all things agile is paramount, and their content is not only comprehensive, it's wonderfully actionable.

—*Grady Booch*
IBM Fellow

The biggest obstacle to overcome in building a high-performance agile team is not learning how, but learning why. Helping teams discover the why is the key to unlock their potential for greater commitment and more creative collaboration. With a focus on values and principles Andrew and Jennifer have provided an outstanding tool to help you and your team discover the why. I can't wait to share it.

—*Todd Webb*
Technical Product Leader at a global e-commerce company

While I was already sold on agile, what I learned from Learning Agile will help my efforts to sell agile across my organization. This book provides more than "just" an engaging way to gain a deep understanding of agile's principles and practices. The easily relatable stories will help make agile compelling to members across your team, so you can begin reaping its rewards.

—*Mark Denovich*
Senior Business Consultant and Head of US development, Centriq Group

An excellent guide for any team member looking to deepen their understanding of agile. Stellman and Greene cover agile values and practices with an extremely clear and engaging writing style. The humor, examples, and clever metaphors offer a refreshing delivery. But where the book really shines is how it pinpoints frequent problems with agile teams, and offers practical advice on how to move forward to achieve deeper results.

—*Matthew Dundas*
CTO, Katori

As an engineer, I always thought the problems that Agile practices help to solve are a direct hit for the industry. As it turns out, becoming Agile is hard; it's more than just the practices. A piecemeal approach to Agile gives, as the the authors call it, "better-than-not-doing-it" results. If you are just getting started, or Agile is only "better-than-not-doing-it", Andrew and Jennifer have a lot of practical advice on how to read between the lines of the Agile Manifesto and really become Agile.

—James W Grenning
Founder of Wingman Software and co-author of the Agile Manifesto

Andrew Stellman and Jennifer Greene have done an impressive job putting together a comprehensive, practical resource that is easily accessible for anyone who is trying to 'get' Agile. They cover a lot of ground in Learning Agile, and have taken great care to go beyond simply detailing the behaviors most should expect of Agile teams. In exploring different elements of Agile, the authors present not just the standard practices and desired results, but also common misconceptions, and the positive and negative results they may bring. The authors also explore how specific practices and behaviors might impact individuals in different roles. This book is a great resource for new and experienced Agile practitioners alike.

—Dave Prior PMP CST PMI-ACP
Agile Consultant and Trainer

If you want to learn about any of the specific approaches to agile, you need to read the specific relevant books. That means you know what you want to do in advance. Not very agile of you, is it? What Andrew and Jenny have done is create an approachable, relatable, understandable compendium of what agile is. You don't have to decide in advance what your agile approach is. You can read about all of them, and then decide. On your way, you can learn the system of agile and how it works.

—Johanna Rothman
Author and Consultant, *www.jrothman.com*

The culture of a software development team often has a greater impact than their expertise or tools do on the success of their project. Stellman and Greene's advice on how to transform an assortment of fragmented individual perspectives into a collaborative unit with shared values and practices should help any software manager regardless of the organization's official methodology. Their comparison of Scrum, XP, Lean, and Kanban techniques analyze the many ways in which Agile principles can be applied. The entertaining case studies illustrate the human dilemmas—and the rewards—of learning to become Agile.

—Patricia Ensworth
President, Harborlight Management Services LLC

Learning Agile is thorough, approachable, practical, and interesting. The values, principles, and methodologies explained in this book are thought-provoking and illuminating, and I look forward to applying them to my team's work.

—Sam Kass
Software architect and tech lead in the financial sector

Andrew Stellman and Jennifer Greene have been there, seen that, bought the T-Shirt, and now written the book! This is a truly fantastic introduction to the major Agile methodologies for software professionals of all levels and disciplines. It will help you understand the common pitfalls faced by development teams, and learn how to avoid them.

—Adam Reeve
Engineer and team lead at a major social networking site

Learning Agile

Andrew Stellman and Jennifer Greene

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Learning Agile
by Andrew Stellman and Jennifer Greene

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com* .

Editor: Mary Treseler
Production Editor: Nicole Shelby
Copyeditor: Jasmine Kwityn
Proofreader: Rachel Monaghan
Indexer: Judy McConville
Interior Designer: David Futato
Cover Designer: Ellie Volckhausen
Illustrators: Andrew Stellman and Jennifer Greene

November 2014: First Edition

Revision History for the First Edition
2014-11-03: First Release
2016-02-05: Second Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449331924* for release details.

978-1-449-33192-4

[LSI]

For Nisha and Lisa,
who have been very patient with us

Table of Contents

Foreword

It seems that people always need something to debate. Was Van Halen better with David Lee Roth or Sammy Hagar? Pepsi or Coke? Lennon or McCartney? Cats or dogs? One such debate in the early days of agile was principles versus practices. Early agilists agreed on a set of principles enshrined in the Agile Manifesto, and many practices were shared across multiple agile approaches. However, there was fierce debate about whether a team should start by understanding the principles of agile software development or whether they should begin by performing the practices even before developing a deep understanding of why.

Proponents of starting with practices took a wax-on/wax-off view of the world. If a team were to act agile, they would be agile. By going through the motions of being agile—pair programming, automating tests and builds, using iterations, working closely with a key stakeholder, and so on—a team would gradually develop an understanding of the principles of agile.

Proponents of starting with principles, on the other hand, contended that practices without principles were hollow. Going through the motions without understanding why did not lead to agility. Agility was (and still is) a focus on continuous improvement. The argument went that a team could not improve continuously if they didn't understand why they were doing the things they were doing.

In *Learning Agile*, Andrew Stellman and Jennifer Greene do the best job I've encountered of stressing the principles of agile without de-emphasizing its practices. They point out that following practices without knowing why is likely to lead only to what they call a "better-than-not-doing-it" level of success. That is, implementing practices alone is helpful, but it falls far short of the true promise of what becoming agile can truly deliver.

I first met Andrew and Jennifer six years ago when they interviewed me for their *Beautiful Teams* book. Although that book does not include *agile* in its title, in many ways the book was about agile. A team that has embraced the principles of agile, mastered the practices it needs, and discarded practices it found unnecessary is truly a

beautiful team. In *Learning Agile*, Andrew and Jennifer focus their discussion on agile by concentrating on today's three most common agile methods—Scrum, Extreme Programming, and Kanban. You'll see how their shared principles have led to different practices within each approach. For example, if you've wondered why Scrum requires end-of-sprint retrospectives but Extreme Programming does not, you'll find the answer here.

In joining Andrew and Jennifer through their exploration of Scrum, Extreme Programming, Lean, and Kanban, you'll read lots of stories. This makes sense—after all, a common practice for many agile teams is telling user stories to describe what a system's users want. You'll meet teams struggling to build the right functionality, taking too long to deliver last year's requirements, mistaking agile for just another form of command-and-control management, being whipsawed by change rather than embracing it, and more. More importantly, you'll read how those teams overcame those problems and how you can, too.

Learning Agile puts an end, once and for all, to the question of which should come first, practices or principles. Its engaging stories and discussions illustrate the simple truth that in agile there can be no separation between principles and practices. In these pages, you'll gain a deeper understanding of how to get started (or get back on track) on your journey to becoming a genuinely beautiful team.

Mike Cohn

Author, *Succeeding with Agile*

Boulder, Colorado

Preface

Acknowledgments

We wrote this book to help you learn agile—but we didn't do it alone. First and foremost, we want to thank our fantastic editor, Mary Treseler. She championed this project from the day we first discussed it with her in an Indian restaurant in downtown Manhattan all the way through to the finished book that you're reading today. She's been an important part of everything we've done with O'Reilly, and we couldn't have done this without her.

We'd also like to thank other friends at O'Reilly, without whom this would not be possible: Mike Hendrickson, Laurie Petrycki, Tim O'Reilly, Ally MacDonald, Andy Oram, Nicole Shelby, and especially Marsee Henon, Sara Peyton, Kathryn Barrett, and all of the fantastic press and PR folks in Sebastopol.

We want to thank Mike Cohn for his wonderful foreword, as well as the really good advice he's given us over the years. We also want to thank him for his great books, because we learned a lot from them! We'd also like to thank David Anderson for giving us some really excellent feedback on Chapters 8 and 9. We want to thank Grady Booch, Scott Ambler, James Grenning, Scott Berkun, Steve McConnell, Karl Wiegers, Johanna Rothman, Patricia Ensworth, Tommy Tarka, Keoki Andrus, Neil Siegel, Karl Fogel, and Auke Jilderda for giving us some really good material and input over the years, especially in *Beautiful Teams*—and especially Barry Boehm, not only for contributing a really fantastic story to that book, but more importantly for laying the intellectual groundwork that much of agile is built on. And we want to thank Kent Beck, Alistair Cockburn, Ken Schwaber, Jeff Sutherland, Ron Jeffries, Tom Poppendieck, Mary Poppendieck, Lyssa Adkins, and Jim Highsmith for their groundbreaking work in agile. We literally would not have been able to write this book without them.

We also want to thank all of our technical reviewers for their excellent feedback and thorough review: Faisal Jawdat, Adam Reeve, Anjanette Randolph, Samuel Weiler, Dave Prior, Randy DeFauw, Todd Webb, Michael DeWitt, and Paul Ellarby.

And finally, we'd like to thank the hundreds of software team members who have been kind enough to talk to us about their problems, solutions, stories, and experiences over the years.

Andrew would like to thank Lisa Kellner. He'd also like to thank everyone in the Computer Science department at Carnegie Mellon who helped him learn some really important ideas, especially Bob Harper, Mark Stehlik, and Randy Bryant. He'd like to thank Tony Visconti, who's been a true mentor and a friend over the years. He'd like to thank his friends Sara Landeau, Greg Gassman, Juline Koken, Kristeen Young, and Casey Dunmore—thinking back, it's kind of amazing how much he's learned about teamwork from these great musicians. He'd also like to thank some really fantastic teammates he's had over his career, including Dan Faltyn, Ned Robinson, Debra Herschmann, Mary Gensheimer, Lana Sze, Warren Pearson, Bill DiPierre, Jonathan Weinberg, and Irene O'Brien. And last but not least, he'd like to thank the two best software teams he's ever worked with, especially Mark Denovich, Eric Renkey, and Chris Winters from Optiron, and Mike Hickin, Nick Lai, Sunanda Bera, and Rituraj Deb Nath from Bank of America.

Jenny would like to thank Nisha Sondhe. She'd also like to thank Christopher Wenger, Brian Romeo, LaToya Jordan, Mazz Swift, Rekha Malhotra, Courtney Nelson, Anjanette Randolph, Shona McCarthy, Ethan Hill, Yeidy Rodriguez, Kyle Mosier, Achinta McDaniel, Jaikaran Sawhny, and Kit Cole for all of their support, laughter, and shenanigans. She'd like to thank her family for their patience and encouragement during the nearly three years this book was under construction. She'd like to thank Tanya and Dilan Desai for their support and help. Finally, she'd like to thank the many colleagues she's learned from over the years. There are too many people who deserve thanks to list them all, but here are a few: Joe Madia, Paul Oakes, Jonathan Weinberg, Bianka Buschbeck, Thor List, Oleg Fishel, Brian Duperrouzel, Dave Murdock, Flora Chen, Danny Wunder, David San Filippo, and Rasko Ristic.

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-

Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://bit.ly/learning-agile*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

CHAPTER 1
Learning Agile

The most important attitude that can be formed is that of desire to go on learning.
—John Dewey, *Experience and Education*

It's an exciting time to be agile! For the first time, our industry has found a real, sustainable way to solve problems that generations of software development teams have been struggling with. Here are just a few of the solutions that agile promises:

- Agile projects come in on time, which is great for teams that have struggled with projects delivered very late and far over budget.
- Agile projects deliver high-quality software, which is a big change for teams that have been bogged down by buggy, inefficient software.
- Code built by agile teams is well constructed and highly maintainable, which should be a relief to teams accustomed to maintaining convoluted and tangled spaghetti code.
- Agile teams make their users happy, which is a huge change from software that fails to deliver value to the users.
- Best of all, developers on an effective agile team find themselves working normal hours, and are able to spend their nights and weekends with their friends and families—possibly for the first time in their careers.

Agile is popular because many teams that have "gone agile" report great results: they build better software, work together better, satisfy their users, and do it all while having a much more relaxed and enjoyable working environment. Some agile teams seem to have finally made headway in fixing problems that have vexed software teams for decades. So how do great teams use agile to build better software? More specifically, how can *you* use agile to get results like this?

In this book, you will learn about the two most popular agile methodologies, Scrum and Extreme Programming (XP). You'll also learn about Lean and Kanban, and how they help you understand the way you build software today and help you evolve to a better state tomorrow. You'll learn that while these four agile schools of thought focus on different areas of software development, they all have one important thing in common: they focus on **changing your team's mindset**.

It's that mindset shift that takes a team from superficially adding a few token agile practices to one that has genuinely improved the way it builds software. The goal of this book is to help you learn both sides of agile: the practices that make up the day-to-day work, and the values and principles that help you and your team fundamentally change the way that you think about building software.

What Is Agile?

Agile is a **set of methods and methodologies** that help your team to think more effectively, work more efficiently, and make better decisions.

These methods and methodologies address all of the areas of traditional software engineering, including project management, software design and architecture, and process improvement. Each of those methods and methodologies consists of **practices** that are streamlined and optimized to make them as easy as possible to adopt.

Agile is *also* a **mindset**, because the right mindset can make a big difference in how effectively a team uses the practices. This mindset helps people on a team share information with one another, so that they can make important project decisions together —instead of having a manager who makes all of those decisions alone. An agile mindset is about opening up planning, design, and process improvement to the entire team. An agile team uses practices in a way where everyone shares the same information, and each person on the team has a say in how the practices are applied.

The reality of agile for many teams that have not had as much success is quite different from its promise, and the key to that difference is often the mindset the team brings to each project. The majority of companies that build software have experimented with agile, and while many of them have found success, some teams have gotten less-than-stellar results. They've achieved some improvement in how they run their projects—enough to make the effort to adopt agile worth it—but they haven't seen the substantial changes that they feel agile promised them. This is what that mindset shift is all about; "going agile" means helping the team find an effective mindset.

But what does "mindset shift" really mean? If you're on a software team, then what you do every day is plan, design, build, and ship software. What does "mindset" have to do with that? As it turns out, the practices you use to do your everyday work depend a lot on the attitude that you and your teammates have toward them.

Here's an example. One of the most common agile practices that teams adopt is called the **daily standup**, a meeting during which team members talk about what they're working on and their challenges. The meeting is kept short by making everyone stand for the duration. Many teams have had a lot of success adding a daily standup to their projects.

So imagine that a project manager is just learning about agile, and wants to add the daily standup to his project. To his surprise, not everyone on his team is as excited about this new practice as he is. One of his developers is angry that he's even suggesting that they add a new meeting, and seems to feel insulted by the idea of attending a meeting every day where he's asked prying questions about his day-to-day work.

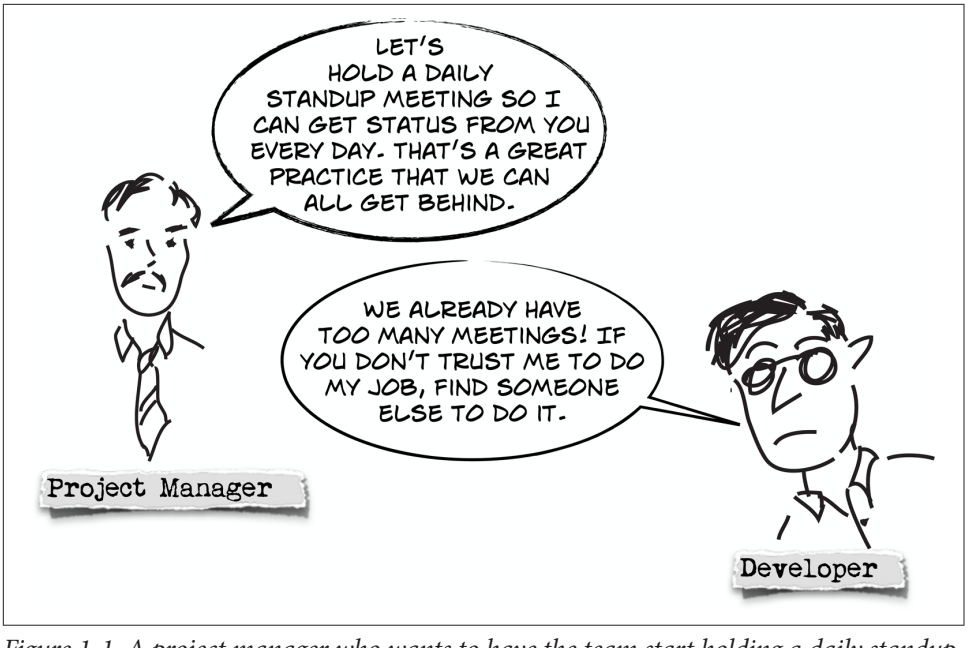

Figure 1-1. A project manager who wants to have the team start holding a daily standup is surprised that not everyone is immediately on board with it.

So what's going on here? Is the developer being irrational? Is the project manager being too demanding? Why is this simple, well-accepted practice causing a conflict?

Both the project manager and the developer have different—and valid—points of view. One of this project manager's biggest challenges is that he puts a lot of effort into planning the project, but when the team runs into problems building the software they deviate from the plan. He has to work very hard to stay on top of everyone on the team, so that he can make adjustments to the plan and help them deal with their problems.

The developer, on the other hand, feels like he's interrupted several times a day for meetings, which makes it very hard for him to get his work done. He already knows what he needs to do to get his own code built, and he doesn't need another person nagging him about plans and changes. He just wants to be left alone to code, and the last thing he wants is yet another meeting.

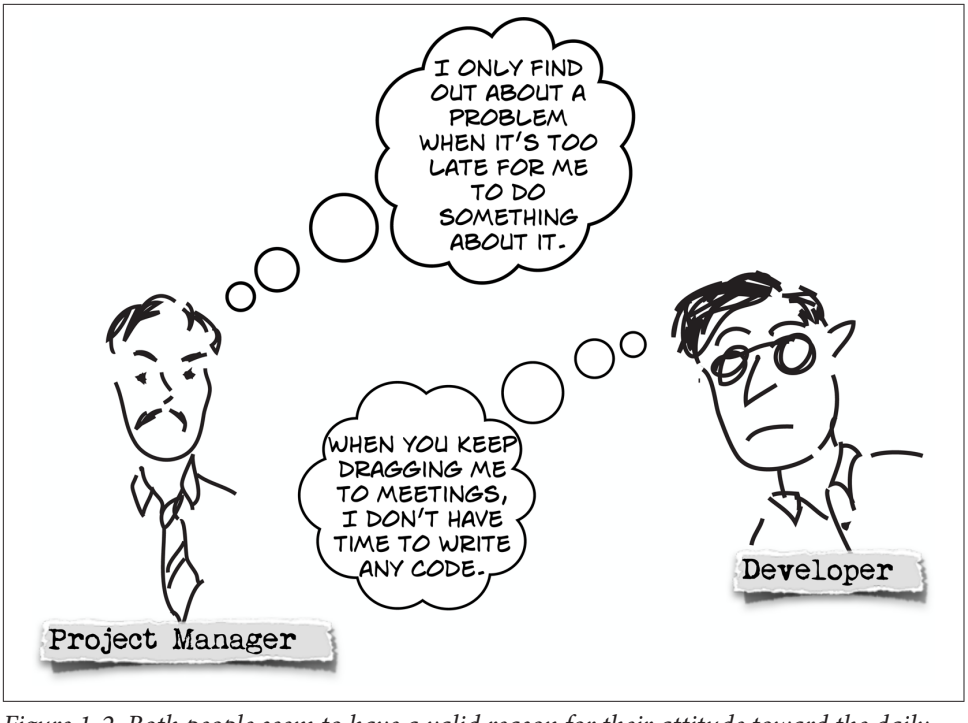

Figure 1-2. Both people seem to have a valid reason for their attitude toward the daily standup meeting. What effect will this have on the project?

Now imagine that the project manager is able to get everyone—even this reluctant developer—to start attending a daily standup meeting. What will that meeting look like? The project manager will be thinking mainly about how people are deviating from his plan, so he'll concentrate on getting status from each person. The developer, on the other hand, wants the meeting to end as quickly as possible, so he'll tune out everyone else while he's waiting to give his update, then say as little as possible when it's his turn, and hope that the whole thing ends quickly.

Let's be clear: this is how many daily standups are run, and while it's not optimal, a daily standup that's run this way *will still produce results*. The project manager will find out about problems with his plan, and the developer will benefit in the long run because those problems that *do* affect him can be taken care of sooner rather than

later—and the whole practice generally saves the team more time and effort than it costs. That makes it worth doing.

But what would happen *if the developer and the project manager had a different mindset*? What if each person on the team approached the daily standup with an entirely different attitude?

For example, what would happen if the project manager felt like everyone on the team worked together to plan the project? Then the project manager would genuinely listen to each team member, not just to find out how they've deviated from *his* plan, but to understand how the plan *that everyone on the team worked together to create* might need to change. Instead of dictating a plan, handing it to the team, and measuring how well the team is following it, he's now working with the team to figure out the best way to approach the project—and the daily standup becomes a way to work together to make sure everyone is doing the most effective thing possible at any given time. As the facts of the project change each day, the team uses the daily standup to work together to make the most effective decisions they can. And because the team meets every day, changes that they discover in the meeting can be put into effect immediately so they don't waste a lot of time and effort going down the wrong path.

And what if the developer felt like this meeting wasn't just about giving status, but about understanding how the project is going, and coming together every day to find ways that everyone can work better? Then the daily standup *becomes important to him*. A good developer almost always has opinions not just about his own code, but about the whole direction of the project. The daily standup becomes his way to make sure that the project is run in a sensible, efficient way—and he knows that in the long run that will make his job of coding more rewarding, because the rest of the project is being run well. And he knows that when he brings up a problem with the plan during the meeting, everyone will listen, and the project will run better because of it.

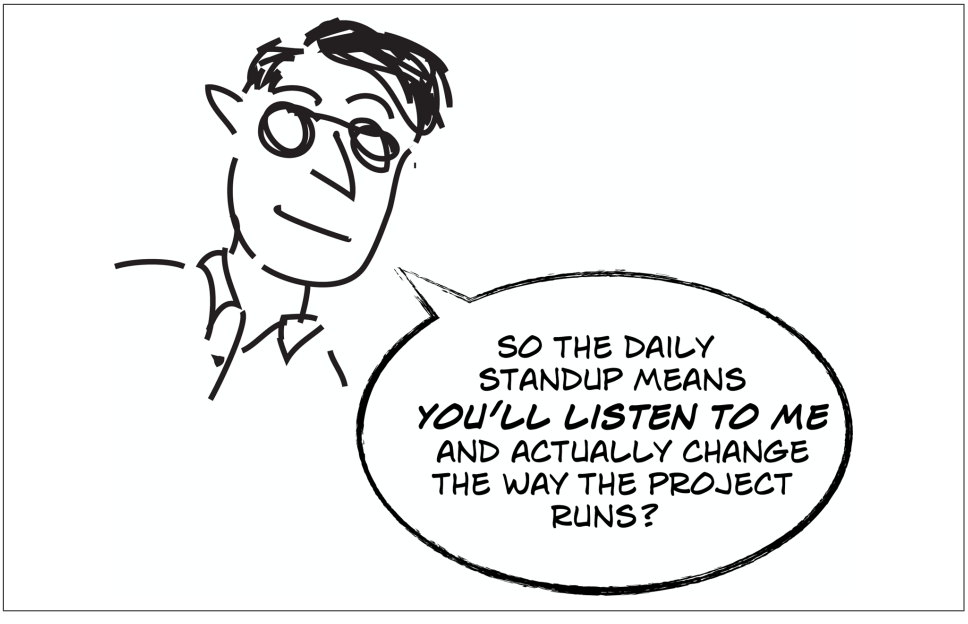

Figure 1-3. When every person on the team feels like they have an equal hand in planning and running the project, the daily standup becomes more valuable—and much more effective.

In other words, if people on the team are in the mindset that the daily standup is simply a status meeting that must be endured, it's still worth doing, but it's only slightly more effective than a traditional status meeting. But if everyone on the team shares the mindset that the daily standup is their way of making sure that everyone is on track, they're all working toward the same goal, and they all have a say in how the project is run, it becomes much more effective—and also more satisfying. The developer has the attitude that this meeting helps him and his team in the long run. The project manager has the attitude that when each person on the team can contribute to the plan, the project gets better results. And when everyone shares these attitudes, the daily standup helps them all work faster, communicate more directly, and get things done more easily.

This is just one small example of how the mindset and attitude of the team have a big effect on how well they'll adopt agile practices. An important goal of this book is to help you understand how your own team's mindset affects your projects and your own agile adoption. By exploring Scrum, XP, Lean, and Kanban, you will learn both sides of the agile coin—principles and practices—and how they come together to help you build better software.

Who Should Read This Book

Do any of these scenarios describe you and your team?

You tried an agile practice, but it didn't really work out. Maybe you implemented daily standup meetings, and now your team meets every day—but you still get blindsided by problems and miss deadlines. Or you started writing user stories and reviewing them with your team and stakeholders, but your developers still find themselves dealing with just as many last-minute changes to add extra features that continue to pop up. Or maybe your team tried to go agile wholesale by adopting a methodology like Scrum or XP, but it seems somehow "empty"—like everyone is going through the "required" motions, but your projects are only marginally improving.

Or maybe you haven't tried agile yet, but you recognize that your team is facing serious challenges, and you don't know where to start. You're hoping that agile will help you with those demanding users who constantly change their minds. Each change your users make requires more work for your team, and leads to "duct tape and paperclips" spaghetti code solutions that make the software increasingly fragile and unmaintainable. It could be that your projects are simply controlled chaos; the primary way software is delivered is through long hours and personal heroics, and you think that agile may offer your team a way out.

What if you're an executive who's worried that teams working on important projects will fail to deliver? Maybe you've heard about agile, but you don't really know what it means. Can you simply tell your team to adopt agile? Or will you need to change your own mindset along with the team?

If any of those situations is familiar to you, and you want to improve how your team works, this book will help.

We explain the agile methodologies: why they're designed the way they are, what problems they address, and the values, principles, and ideas that they embody. By giving you the "why" in addition to the "how," we'll help you to recognize the principles that apply to the particular development problems specific to your team, company, and projects. And we'll show you how to use that information to guide your choice of methodologies and practices.

There's one other sort of person we wrote this book for: the **agile coach**. Teams and companies are increasingly relying on agile coaches to guide them in adopting agile methodologies and practices, and to get each team member into the right mindset. If you're an agile coach, we will give you tools to help you better communicate these ideas to your team, and to overcome some of the challenges that you face every day when helping a team become more agile.

Our Goals for This Book

What we want for you:

- We want you to understand the ideas that drive effective agile teams, and the values and principles that bring them together.
- We want you to understand the most popular agile schools of thought—Scrum, XP, Lean, and Kanban—and how they can all be agile, even though they're very different from each other.
- We want to teach you specific agile practices that you can apply to your projects today—but we also want to give you the framework of values and principles that you'll need to implement them effectively.
- We want to help you understand your own team and company better, so that you can choose an agile approach that matches your mindset (or comes as close as possible)—but also help you and your team start to learn a new way of thinking that will help you become a more effective agile team.

How do all the different agile methodologies and practices give you better software? Why do they give your team the ability to handle change better? Why are these things agile? Does it really matter that you're using, say, index cards for planning, or that you're standing up during meetings? These questions are difficult and often confusing for people just starting their path to agility. By the end of this book, you'll be able to answer them for yourself.

If you look at many of the blogs and articles out there discussing agile software development, one of the first things you'll read is that "Agile is good, and waterfall is bad." Why is agile "good," and waterfall "bad"? Why are they in conflict with each other? Is it possible to work on a team that follows a waterfall process and still be agile? By the end of this book, you'll able to answer those questions, too.

Getting Agile into Your Brain by Any Means Necessary

This book is called *Learning Agile* because we *really* want you to learn agile. We've spent the last 20+ years working with real teams building real software for real users day in and day out. We've also spent the last 10+ years writing books about building software (including two very successful books in the O'Reilly *Head First* series about managing projects and learning to code). This experience has helped us find many different ways to get complex and technical ideas into your brain without boring you to death.

We've done our best to take this material and make it as interesting and engaging as possible...but we need your help to do it. Here are the tools and techniques you'll find throughout this book to help make these ideas stick in your head:

Narratives

Denoted by the  icon. Think about the last technical book that you read. Can you remember all the major topics it covered, and in what order? Probably not. Now think about the last movie you saw. Can you remember the major plot points, and the order in which they happened? Almost certainly. That's because your brain is wired to remember things that cause you to have an emotional reaction. In this book, we use that to our advantage. We'll use narratives—with people, dialog, and conflict—to show you what it looks like when real people encounter agile. These people will run into problems.

What we need from you: Try to imagine yourself in their situation, because that will give you an emotional connection to these ideas, and make it easier for you to remember and understand them. Keep an open mind about these narratives, especially if you're the sort of learner who doesn't like to read fiction. There's real learning in each narrative, and they're part of the core material in this book.

Illustrations

Different people learn in different ways. Some people are more visual learners, and ideas will "click" much more easily for them when they see a picture. We want to give you as many tools as possible to learn, so we included many illustrations throughout this book. In some cases, we rely heavily on visual metaphor, like using geometric shapes to represent different features, or gears to represent complex software.

What we need from you: If you're not a visual learner, some of the illustrations may seem superfluous at first, and you may find yourself thinking that a specific illustration doesn't make sense. This is a good learning opportunity for you, and if this happens then you should take a minute and try to figure out what someone who *is* a visual learner might get from the illustration. That will help you gain a deeper understanding of the concept.

Redundancy

Most technical books introduce an idea, describe it completely, and then move on to the next one. That's an effective way to get as much information into a book as possible, but that's not how our brains work. Sometimes you need to see the same concept more than once before you get that "aha!" moment. This is why we will sometimes come back to the same concept several times within a chapter, or in different chapters. **This redundancy is intentional**—we do it to help you get to that "aha!" moment.

What we need from you: When you run across a concept for the second or third time, you might be tempted to think, "Didn't they already cover this?" Yes, we did, and it's really good that you noticed it! There are other readers who didn't

notice it, the same way you probably won't notice every time we use redundancy. It's all done to help you learn.

Simplification (at first!)

Sometimes it's easier to understand a complex subject by first just scratching the surface, and letting that sink in. We do that many times in this book, introducing a simplified (but still technically correct!) version of a concept, and elaborating on it later. This works on two levels. If it's an idea that you already understand in depth, then you'll recognize the simplification and react to it emotionally—and that keeps you engaged. On the other hand, if you aren't aware of the concept, it helps give you a gentler introduction to prepare you for the more in-depth description later on.

What we need from you: If you feel like something is oversimplified, don't just dismiss it, and definitely don't assume that we missed a major idea or that we glossed over or forgot an important point. Chances are, you'll find the point that you're looking for later on in the book. If it helps, think of a simplified introduction of a difficult concept as a sort of "Hello, World!" program for your brain—it gives readers who aren't familiar with the concept a feeling of encouragement so that they know they're on the right track, and it sets the stage for a deeper understanding later.

Conversational/casual tone

We keep a casual tone throughout this book in order to keep the material as engaging as possible. We use humor and occasional cultural references, and will sometimes speak directly to you or refer to ourselves using pronouns like "us" and "you." There's actually science behind this—cognitive research[1] that shows that your brain remembers more when you feel like you're in a conversation.

What we need from you: While most people have no problem with a casual tone, some people really dislike it. For example, some readers bristle every time they see a contraction. For others, a casual tone makes them feel like the book isn't authoritative enough. We understand that. Believe it or not, you'll actually get used to it faster than you might think.

Key Points

Denoted by the  icon. Throughout each chapter we'll summarize key points that were covered recently. This helps you make sure that you "got" everything, and that you didn't miss an important concept. It also gives your brain a quick break from learning.

1 If you're interested in learning more about how conversational tone helps you learn, have a look at *E-Learning and the Science of Instruction* by Ruth C. Clark and Richard E. Mayer (Wiley, 2011).

What we need from you: Don't just skip over the Key Points sections. Take a minute and look through them. Do you remember each of those points? If not, don't be afraid to flip back a few pages and refresh your memory.

Frequently Asked Questions

Denoted by the  icon. We spend most of our time actually working on software teams building real software for real users, but we've also spent years giving talks and presentations about agile, and talking to many, many people. And over all of that discussion, there are some questions that people kept asking over and over again.

What we need from you: Read the Frequently Asked Questions at the end of each chapter. Was this a question that you had? If so, do you like the answer? You may not always like the answer, but try to find the truth in it. If it wasn't a question that you had, try to figure out why someone might ask it—that can help you see the material from a different perspective (and you'll learn in Chapter 2 why that is important for a team).

What You Can Do Today

Denoted by the  icon. The most effective way to learn something is to do it! At the end of each chapter, we include a short section that gives you a few things that you can do today, right now, either alone or with your team.

What we need from you: Obviously, the best thing that you can do is actually try those things! But the reality is that not every team or company is open to this—and one of the most important things you'll learn throughout the book is that trying to put a practice in place on a team with an incompatible mindset can end badly. So before you try these things, think about how your team will respond. That can be as effective a learning tool as actually doing it.

Where You Can Learn More

Denoted by the  icon. Isaac Newton once said, "If I have seen further it is by standing on the shoulders of giants." We're lucky to be writing this at a time when there have been so many groundbreaking books written about agile software development. After each chapter, we'll give you a few places where you can learn more about what we just taught.

What we need from you: Keep learning! This book is a thorough overview of Scrum, XP, Lean, and Kanban, but we can't cover every detail of each of these ideas. We didn't come up with most of the ideas in this book; luckily, you can learn from the people who did.

Coaching Tips

Denoted by the  icon. An agile coach is someone who helps teams learn agile. This book is written for someone who is learning agile, but it can also be used as a guide to help an experienced agile coach introduce these ideas to her team. If you're an agile coach, look for the coaching tips at the end of each chapter. They'll help you take the ideas and the approach that we use and adapt them to your own team.

What we need from you: Even if you're not a coach, you should still read the coaching tips. That's because one effective way to learn is to put yourself in the shoes of someone who is teaching others. If you're learning these concepts for the first time, imagine how you might use these coaching tips to help your team learn more about agile.

How This Book Is Structured

This book is structured to help you understand agile by teaching the values and principles of an effective software team, the schools of thought that embody those values, and the practices that make them up.

The next two chapters will help you understand the values and principles that will help you to adopt an agile mindset. They will give you the tools to figure out if your team and your company are ready to adopt agile, which parts of agile will resonate with your team, and which might be more difficult to implement:

- Chapter 2, *Understanding Agile Values*, describes the core agile values. We will show you an example of a team struggling with a software project, and help you recognize that a main source of the struggle is a "fractured perspective." We'll explain the agile values, and use a metaphor to help you see how those values bring the team's perspective together.
- Chapter 3, *The Agile Principles*, describes the principles that agile teams use to make decisions about how to run their projects. We'll show you the purpose and idea behind each of the principles, illustrating them with a practical example from a software project.

The following six chapters will teach you about the most popular agile schools of thought: Scrum, XP, Lean, and Kanban. You'll learn their basic application in a way that you can put in place on your team today:

- Chapter 4, *Scrum and Self-Organizing Teams*, uses Scrum, a popular agile methodology, to teach you how self-organizing teams work. We'll give you advice for adopting Scrum on your own projects, and tools to help your team learn to self-organize.

- Chapter 5, *Scrum Planning and Collective Commitment*, shows you specific practices that Scrum teams use to plan their projects, and how those practices help your team commit as a whole to delivering valuable software. We'll show you how real-life adoption of Scrum depends on how well the Scrum values match the culture of your team and your company, and what to do if they don't.
- Chapter 6, *XP and Embracing Change*, teaches you about the primary practices of Extreme Programming, and the XP values and principles. You'll learn how they help get each team member into the right mindset to build better code: instead of hating change, every person on the team learns to embrace it.
- Chapter 7, *XP, Simplicity, and Incremental Design*, teaches you about the final three primary practices of XP, and how they help you avoid serious code and design problems. You'll learn how all of the XP practices form an ecosystem that leads to better, more maintainable, flexible, and changeable code.
- Chapter 8, *Lean, Eliminating Waste, and Seeing the Whole*, teaches you about Lean, and the values that help you get into the lean thinking mindset. And we show you how the Lean thinking tools can help your team identify waste and eliminate it, and see the whole system that you use to build software.
- Chapter 9, *Kanban, Flow, and Constantly Improving*, teaches you about Kanban, its principles and how they relate to Lean, and its practices. You'll learn how its emphasis on flow and queuing theory can help your team put lean thinking into practice. And you'll learn about how Kanban can help your team to create a culture of continuous improvement.

The world of agile includes more than mindsets, methodologies, and schools of thought. Companies are increasingly relying on agile coaches to help their teams adopt agile. This is why we included the final chapter in the book:

- Chapter 10, *The Agile Coach*, teaches you about agile coaching: how teams learn, how an agile coach can help a team change their mindset so that they can more easily adopt an agile methodology, and how that coach can help you and your team become more agile.

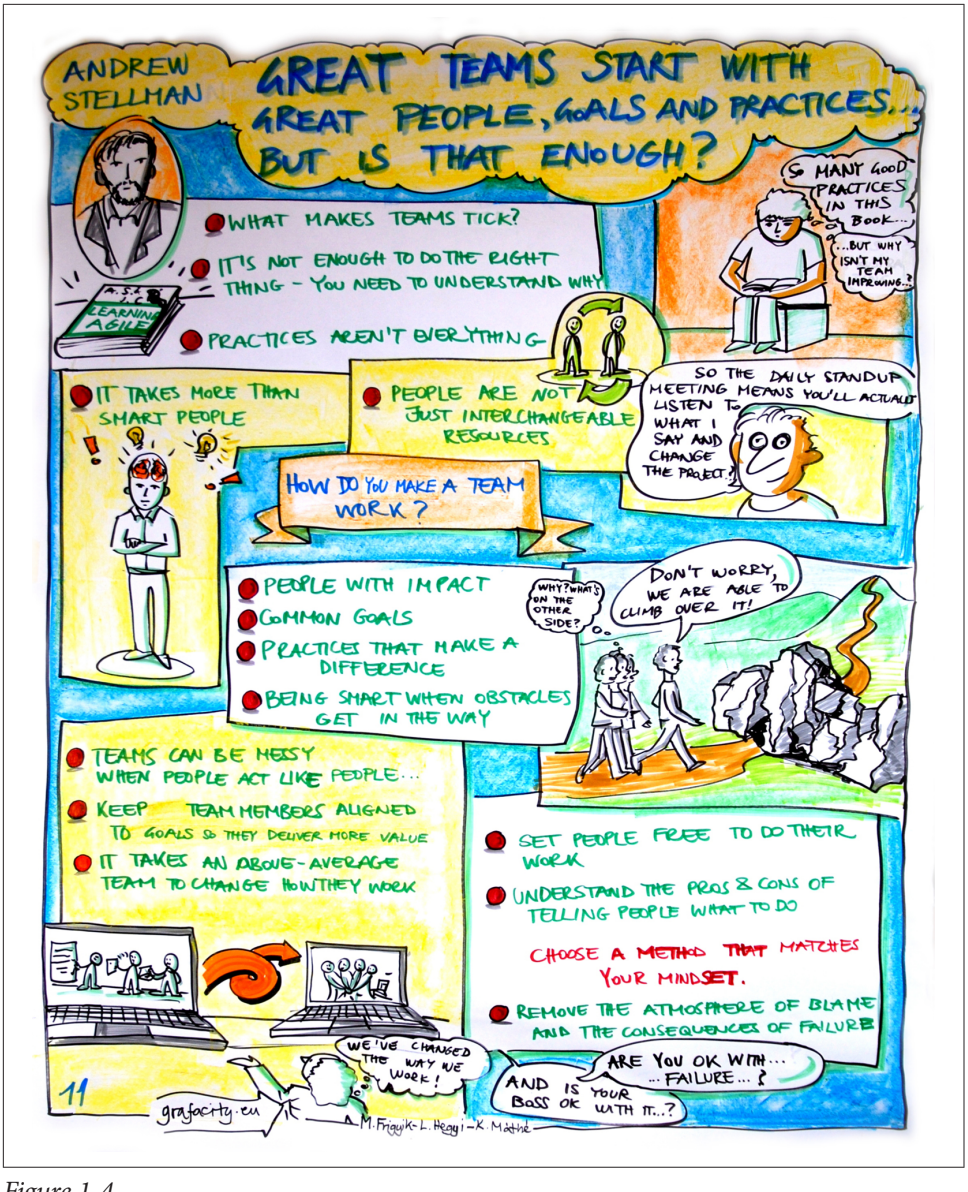

Figure 1-4.
Live graphic recording of Andrew Stellman's talk at the STRETCH 2013 conference in Budapest, Hungary. The talk was based on the material in this chapter.

Graphic recording: Kata Máthé and Márti Frigyik
www.remarker.eu

CHAPTER 2
Understanding Agile Values

We do not act rightly because we have virtue or excellence, but we rather have those because we have acted rightly. We are what we repeatedly do. Excellence, then, is not an act but a habit.

—Aristotle, *Nichomachean Ethics*

Agile, as a movement, is different from any approach to software development that came before it, because it started with ideas, values, and principles that embody a mindset. It's through the lens of these ideas that you can begin to become more agile as a practitioner, and more valuable as a member of your project team.

The agile movement is revolutionizing the world of software development. Teams that adopt agile have consistently reported improvements—sometimes huge leaps—in their ability to build great software. Teams that successfully adopt agile build better, higher quality software products, and they do it faster than before.

Our industry is at a turning point with agile. Agile has gone from being the underdog to becoming an institution. For the first few years of agile, people adopting it struggled to convince their companies and teammates that it worked, and that it was worth doing. Now, there is little question that agile development is a highly effective way to build software. In fact, in 2008, an important survey[1] found that more than half of all software teams surveyed were using agile methodologies, practices, or principles—and agile has only grown since then. And agile teams are increasingly going beyond the problem of how to be agile themselves, and are starting to figure out how to spread agile development throughout their companies.

But it wasn't always like this. Traditionally, companies have used a **waterfall process** when running their software projects, in which the team defines the requirements up

1 The Forrester 2008 Global Agile Company Online Survey

front, plans the project in its entirety, designs the software, and then builds the code and tests the product. Plenty of software—great software, but also lousy software—has been built this way over the years. But as the decades went on, different teams in different companies kept running into the same kinds of problems...and some people started to suspect that a major source of project failure might be the waterfall process itself.

The story of agile started when a small group of innovative people got together to try to find a different way of thinking about these problems. The first thing they did was come up with a set of four core values that are common to successful teams and their projects, which they called the *Manifesto for Agile Software Development*:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

In this chapter, you'll learn about these values—where they came from, what they mean, and how they apply to your project. You'll follow a waterfall-weary team through their first attempt at implementing agile before they really understand how those values apply to them. As you read their story, see if you can spot how a better understanding of these values could have helped them avoid their problems.

Narrative: a team working on a streaming audio jukebox project

Dan – lead developer and architect

Bruce – team lead

Joanna – newly hired project manager

Tom – product owner

A Team Lead, Architect, and Project Manager Walk into a Bar...

Dan is a lead developer and architect at a company that makes coin-op games and kiosks. He's worked on projects ranging from arcade pinball machines to ATMs. For

the last few years, he's been working with a team lead, Bruce. They bonded on a release of the company's biggest product, a Vegas slot machine called the "Slot-o-matic Weekend Warrior."

Joanna was hired a few months ago as a project manager to head up a project building software for a new line of streaming audio jukeboxes, which the company wants to introduce and sell to bars and restaurants. She was a real prize—they poached her from a competitor that already has a successful jukebox on the market. She's been getting along really well with Dan and Bruce, and she's excited about getting started on a new project with them.

Dan and Bruce are much less excited about the new project than Joanna. They all went out for drinks after work one day, and Bruce and Dan started explaining why the team came up with their own name for the slot machine project: the "Slog-o-matic Weekend Killer."

She wasn't happy to learn that in this company, failing projects are the rule, not the exception. The last three projects were declared a success by the company's managers, but they only got out the door because Dan and Bruce worked incredibly long hours. Worse, they held their noses and took shortcuts in the code that are causing support nightmares today—like when they hastily patched up a prototype for one feature and pushed it out into production, and it later turned out to have serious performance problems because pieces of it were never built to scale up.

As they talked, Joanna recognized the pattern, and knew what was causing the problem: the company follows a particularly ineffective *waterfall process*. A team following a waterfall process tries to write down as early as possible a complete description of the software that will be built. Once all of the users, managers, and executives agree on exactly what the software must do (the *requirements*), they can hand a document that contains those requirements (the *specification*) to a development team, and that team will go off and build exactly what's written. Then a testing team comes in afterward, and verifies that the software matches the document. Many agile practitioners refer to this as "big requirements up front" (sometimes abbreviated BRUF).

But Joanna also knew from her own projects over the years that theory often differed from practice, and that while some teams have a very effective waterfall process, many of them struggle with it. She was starting to think that this team might be one of those that struggled.

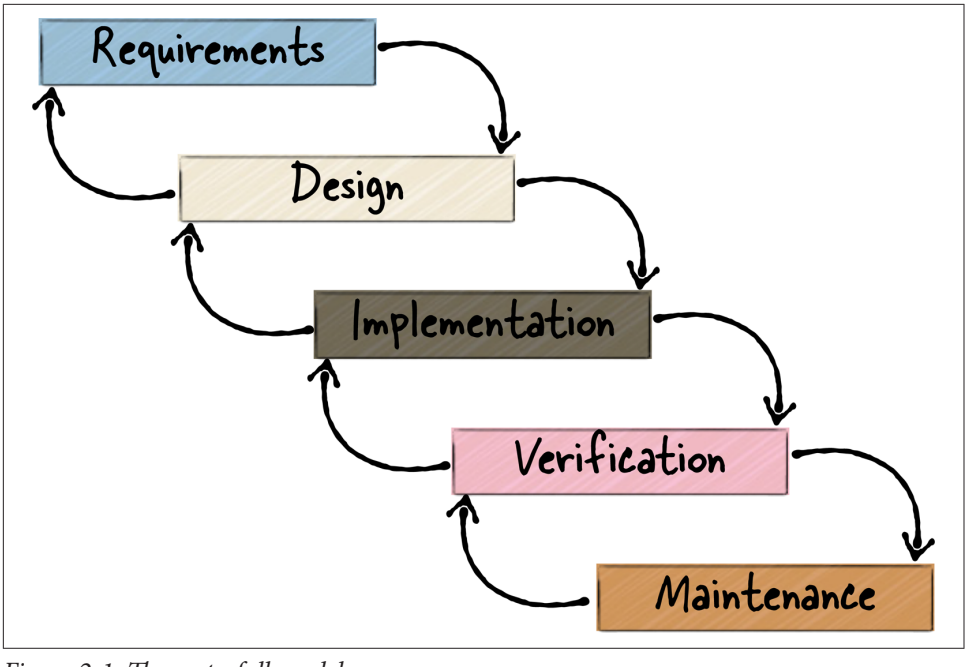

Figure 2-1. The waterfall model.

As they talked, Bruce and Dan confirmed a few things that reinforced her opinion. Just as Joanna suspected, there were a lot of specifications sitting in large binders and gathering dust on shelves all across the company. Somehow, everyone expected a group of users, managers, and executives to create a perfect requirements specification. In real life, the spec had a nasty habit of changing so that it would be inaccurate by the time the team got their hands on it, and would progress to being disastrously wrong by the time the team finished building the software. Bruce, Dan, and a lot of other people at the company knew that it was unreasonable to expect the perfect spec, but they still ran their projects as if it were possible.

As the evening went on, Bruce got more comfortable (and tipsy), and he brought up another problem that had been nagging him: that many teams he'd been on at their company had a lot of trouble actually building their software. Even if the users got the requirements right (which rarely happened), and even when the team understood those written requirements perfectly after reading them (which had yet to happen to this day), they often used inferior tools and had a lot of trouble with software design and architecture. This led to Bruce's teams repeatedly building software that had a lot of bugs, and was often an unmaintainable mess.

Both of these problems led to many failed projects, especially because they considered any project where they had to work many 90-hour weeks and deliver buggy code to be a failure. Joanna explained that the biggest cause of those failures was the *inability*

of the waterfall process the company followed to handle change. In a perfect world, the waterfall process would work just fine, because at the start of the project everyone would know exactly what they'd need at the end. That way, they could write it all down in a neat spec and hand it off to the team to build. But real-life projects never seemed to work out that way.

Dan and Bruce were now officially drunk, and deep into a marathon gripe session with Joanna. Dan told her that on almost every project that they'd worked on, the customers decided partway through that they needed the team to build something different than what they'd originally planned on. Then everyone had to go back to the beginning of the waterfall. According to the strict waterfall process the team was following, they were supposed to write entirely new specifications, come up with a different design, and build a whole new plan. But in reality, this almost never happened, and it was rare that they had time to throw out all of the code that had been written so far. Instead, they usually ended up hacking together a solution out of the existing code. This rework led to bugs, because taking software that was designed for one purpose and hastily modifying it to do something else often results in messy, tangled code—especially when the team is under pressure. Adapting it to the new use would have taken up precious project time, so they ended up with poor workarounds and brittle code.

What Dan, Bruce, and Joanna were starting to realize by the end of the night was that their project's problems were caused by *overly rigid documentation*, *poor communication*, and *bugs*, which led to projects that could not keep up with normal project changes.

At the end of the evening, the bartender called taxis for all three of them. Just before they left, Dan said he was relieved to get a lot of that off of his chest. Joanna was happy to have a better picture of the project that she was joining...but a lot less optimistic. Will she be able to find a way, she wondered, to fix some of these problems?

No Silver Bullet

Today we know that there's no single "best" way to build software. But while that's not a controversial statement now, for much of the last century many people in the industry would have loudly disagreed. There was a general feeling among many practitioners that they could discover a single, highly prescriptive "silver bullet" method that would solve project problems for everyone, everywhere. Many people felt that developers could build software just by following a set of instructions, like following a recipe or assembling a product on an assembly line.

(Ironically, one of the most quoted papers in software engineering is Fred Brooks's 1986 essay, "No Silver Bullet," in which he shows conclusively why this is an impossible goal. That has yet to stop people from trying to find one!)

There were many proposed silver bullet solutions to these kinds of problems. They typically came in one of two forms: a **methodology** that claimed to give teams a foolproof way of building software, or a technology that programmers could use to prevent or eliminate bugs. The idea was that if a company decided on a methodology and a technology, then all the team had to do was to follow the company orthodoxy, and great software would start pouring out.

Dan and Bruce know firsthand that this doesn't work, because they lived through years of managers in their company throwing methodologies and technologies at their projects without any real, lasting improvement. The company's attempts at finding a silver bullet software process were usually enormously disappointing for everyone involved—and especially for Bruce and Dan, because they were asked to follow an ever-changing series of processes that they hadn't asked for.

Joanna was also no stranger to this from her own career. At her last job, she was routinely handed a rigid set of requirements, and ordered to come up with a plan to build them out into software. Teams were then given her plan, and ordered to follow it to the letter. Teams that "planned the work, and worked the plan" were doomed to build software that was often outdated and not useful to their users even on the day that it was deployed.

One thing that gave Joanna pause was that some teams she worked on actually did manage to get great software out the door by following a waterfall (or waterfall-like) process that was heavy on up-front documentation. She'd managed some of her best projects at a company that built software for medical devices using waterfall practices.

Waterfall really can work. In fact, if you actually know what you need up front, then writing it down first is a pretty efficient way to build software. Joanna's medical device software projects were rare examples where the requirements were actually right from the beginning, and needed very few changes during the project.

But it takes more than just stable requirements to run a successful waterfall project, which is why they run into so many problems. Teams that build great software using a waterfall process typically have a few common characteristics:

- Good *communication*, because the teams that were successful in a company that mandated waterfall were the ones that consistently talked to their users, managers, and executives throughout the project.
- Good *practices*, especially ones like code reviews and automated testing, which are aimed at finding bugs as early as possible in the process. They usually called this "defect prevention," and it required teams to actively think about how those bugs got into the code in the first place.
- Drawers full of documentation that have rarely been opened, because the people on the team understand that the act of writing down the plan—and the questions

that get asked during that planning—is more important than mindlessly sticking to it afterward.

There's one other piece of the puzzle. Dan started his career after the 1990s revolution in software development tools and technology, so he's only ever worked on teams that used object-oriented development to build better software designs. Version control systems, automated testing, better integrated development environments (IDEs) that include features to automate coding practices like refactoring and class design, and other revolutionary tools all help Dan keep control of the code. Bruce has been working on projects longer than Dan, and he watched developers on his teams increasingly adopt software tools over the years to automate routine and repetitive tasks. Bruce and Dan know from their own projects that the most successful ones made effective use of these practices, tools, and ideas. That left more time for them to talk to their users and teammates, and to think about the problems they had to solve instead of fighting with the code.

And as it turns out, when waterfall projects are run effectively, it's because their teams take to heart *many of the same values, principles, and practices that agile projects follow*. Projects that are run using some agile techniques and practices—but that don't really follow the agile values and principles—often end up running into the same kinds of problems that plague waterfall projects.

Unfortunately for Bruce, Dan, and Joanna, they're about to learn this the hard way.

Key Points

- A **waterfall process** requires a team to write down a complete description of the software at the beginning of the project, and then build exactly what they wrote.
- The waterfall process made it difficult to respond to change because of the focus on documentation rather than collaboration.
- There is *no silver bullet* process or practice that makes projects run perfectly.
- Teams that make waterfall work do it by *adopting effective software practices and principles*, especially ones that improve communication.

Agile to the Rescue! (Right?)

You probably know what a waterfall process feels like, even if you're just learning the term "waterfall" for the first time.[2] Joanna, Bruce, and Dan do, too. Before jumping into planning the jukebox project, they talked about how waterfall processes had caused problems for their teams in the past.

On their last project, Bruce and Dan worked with Tom, a customer account manager at the company, who spent a lot of time on the road helping customers at arcades, casinos, bars, and restaurants install and use their products. The three of them spent the first few weeks of the project building up a specification for the new slot machine. Tom was only in the office half the time, which gave Bruce and Dan time to start designing the software and planning the architecture. Once all three of them had agreed on the requirements, they called a big meeting with the CEO and senior managers of the company to review the requirements, make necessary changes, and get approval to start building.

At that point, Tom went back out on the road, leaving the work up to Bruce and Dan. They broke the project down into tasks, dividing them up among the team, and had everyone start building the software. When the team was almost done building the software, Tom gathered a group of business users, project managers, and executives into a big conference room to do a demo of the nearly complete Slot-o-matic Weekend Warrior software.

It didn't go nearly as smoothly as they'd expected.

At the demo, they had an awkward conversation where the CEO asked, "The software looks great, but wasn't it supposed have a video poker mode?" That was an unfortunate discovery. Apparently, the CEO was under the impression that they were working on software that could be deployed to either the slot machine hardware or the video poker hardware. There had been a lot of discussion of this between the senior managers, the board, and the owners of their two biggest customers. Too bad nobody had bothered to tell the team.

The worst part about it was that if Dan had known about this earlier in the project, it wouldn't have been difficult to change direction. But now, they had to rip out enormous amounts of code they'd already written, and replace it with code retrofitted from the video poker project. They spent weeks troubleshooting weird bugs that were caused by integration problems. Dan spent many late nights complaining to Bruce that this was 100% predictable, and that this almost always happens when code built

2 If you're a project manager and you've prepared for the PMP exam, you've learned all about a waterfall process. To be fair, that's because a PMP certification spans many different methodologies across many industries, not just software. When you build a skyscraper or a bridge, it's generally a good idea to have a complete set of blueprints up front, even if they do change along the way.