Microsoft

Agile Project Management with Kanban

Best practices

Eric Brechner

Praise for *Agile Project Management with Kanban*

"I have been fortunate to work closely with Eric for many years. In that time he has been one of the most productive, consistent, and efficient engineering leaders at Xbox. His philosophy and approach to software engineering are truly successful."

—Kareem Choudhry, Partner Director of Software Engineering for Xbox

"Eric easily explains why Kanban has proven itself as a useful method for managing and tracking complicated work. Don't expect this book to be an overview, however. Eric channels his deep understanding and experiences using Kanban at Microsoft to help you identify and avoid many of the common difficulties and risks when implementing Kanban."

—Richard Hundhausen, President, Accentient Inc.

"Learning how Xbox uses Kanban on large-scale development of their platform lends real credibility to the validity of the method. Eric Brechner is a hands-on software development management practitioner who tells it like it is—solid, practical, pragmatic advice from someone who does it for a living."

—David J. Anderson, Chairman, Lean Kanban Inc.

"As a software development coach, I continuously search for the perfect reference to pragmatically apply Kanban for continuous software delivery. Finally, my search is over."

—James Waletzky, Partner, Crosslake Technologies

"Kanban has been incredibly effective at helping our team in Xbox manage shifting priorities and requirements in a very demanding environment. The concepts covered in Agile Project Management with Kanban *give us the framework to process our work on a daily basis to give our customers the high-quality results they deserve."*

—Doug Thompson, Principal Program Manager, Xbox Engineering

"An exceptional book for those who want to deliver software with high quality, predictability, and flexibility. Eric's in-depth experience in the software industry has resulted in a realistic book that teaches Kanban in a simple and easy way. It is a must-read for every software professional!"

—Vijay Garg, Senior Program Manager, Xbox Engineering

This page intentionally left blank

Microsoft

Agile Project Management with Kanban

Eric Brechner

With a contribution from James Waletzky

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

Library of Congress Control Number: 2014951864
ISBN: 978-0-7356-9895-6

Printed and bound in the United States of America.

First Printing

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at mspinput@microsoft.com. Please tell us what you think of this book at http://www.microsoft.com/learning/booksurvey.

Acquisitions Editor: Devon Musgrave
Developmental Editor: Devon Musgrave
Project Editor: Devon Musgrave
Editorial Production: Rob Nance, John Pierce, and Carrie Wicks
Copyeditor: John Pierce
Indexer: Lucie Haskins
Cover: Twist Creative • Seattle

Table of Contents

Introduction . . . . . *ix*

Chapter 1 Getting management consent **1**

An open letter to your manager 2
Problem 2
Solution 2
Risks 3
Plan 3
Moving forward 4
Checklist 5

Chapter 2 Kanban quick-start guide **7**

Step 1: Capture your team's high-level routine 7
Step 2: Redecorate your wall 8
Step 3: Set limits on chaos 10
Step 4: Define done 13
Step 5: Run your daily standup 14
Troubleshooting 17
Checklist 24

Chapter 3 Hitting deadlines **25**

Populate your backlog 25
Establish your minimum viable product (MVP) 27
Order work, including technical debt 28
Estimate features and tasks 29
Track expected completion date 31
Right-size your team 33

Basic approach 34
Advanced approach 35
Checklist 37

Chapter 4 Adapting from Waterfall 39

Introducing Kanban to a Waterfall team 39
Working in feature teams 42
Completing features before starting new ones 43
Dealing with specs and bugs 44
Specs 44
Bugs 45
Engaging with customers 46
Celebrating performance improvements 48
Rude Q & A 51
Checklist 56

Chapter 5 Evolving from Scrum 57

Introducing Kanban to a Scrum Team 58
Mapping the roles and terms 60
Evolving the events 61
Celebrating performance improvements 62
Rude Q & A 65
Checklist 70

Chapter 6 Deploying components, apps, and services 71

Continuous integration 72
Continuous push 75
Continuous publishing 77
Continuous deployment 79
Checklist 83

Chapter 7 Using Kanban within large organizations 85

Deriving a backlog from big upfront planning 86

Ordering work based on dependencies 87

Fitting into milestones 91

Communicating status up and out 92

Dealing with late or unstable dependencies 94

- Late dependencies 94
- Unstable dependencies 95

Staying productive during stabilization 98

Checklist 100

Chapter 8 Sustained engineering 101

Define terms, goals, and roles 101

- Consistent vocabulary 102
- Challenges and goals 102
- Define roles and responsibilities 103

Determine SE ownership 104

Lay out support tiers 105

- Tier 1 106
- Tier 2 106
- Tier 3 106

Collaborate for efficiency 106

- Triage 106
- Quick-solve meeting 108

Implement Kanban SE workflow 108

- Escalations 109
- Bugs/Other Work 109

Kanban tools 111

Troubleshooting 112

Checklist 115

Chapter 9 Further resources and beyond 117

Expanding Kanban to new areas of business and life. 117
Scaling Kanban up, down, and out 118
Personal Kanban. 120

Mixing Agile and Lean with Kanban 120

Why Kanban works 123
Single-piece flow 124
Theory of constraints (TOC) 124
Drum-buffer-rope 126

Improving beyond Kanban 128
Critical chain 129
Lean development 130
Global optimization 132

Checklist 136

Index *137*

About the author *145*

Introduction

Dedicated to Corey Ladas, who invited me to see through new eyes.

I'm a professional software developer. I've been one for decades (currently with Xbox). I don't get paid to be a certified process geek. I don't get paid to be an evangelical process zealot. I get paid to deliver software that customers love, with high quality, on time, and at low cost.

I develop software in a highly volatile environment, where priorities, requirements, and expectations change daily. I develop software in a highly competitive environment—for market share among products and for compensation among peers. My software development peers inhabit this same world, regardless of where they work or what products they produce.

I'm always looking for advantages over my competition—ways that make my life easier and more productive while also resulting in better products for my customers. When I was at Bank Leumi, Jet Propulsion Laboratory, Graftek, and Silicon Graphics in the 1980s, I focused on what we now refer to as design patterns and unit testing. During the 1990s, while I was at Boeing and starting out at Microsoft, my teams and I tried Waterfall milestones and stabilization periods of different durations, T-shirt estimation, asserts, bug jail, continuous integration, and design and code reviews. In the 2000s, the Microsoft teams I managed experimented with Team Software Process, Scrum, code inspection, static analysis, planning poker, pair programming, and test-driven development. Now in the 2010s, I've found continuous deployment, Kanban, and a little nirvana.

Some of the methods I just listed may not be familiar to you. Most of the professional software developers I've known don't like experimenting with how they do their jobs. They find an approach that works for them, usually the one they learn from their first professional software team, and tend to stay with that approach.

Trying different methodologies is painful and has an initial drain on productivity, but it has enabled my teams to outperform those of my peers. My teams are significantly smaller than other teams doing similar work, yet they produce significantly more value at significantly higher quality in the same or less time. That's not because I've filled my teams with mythical developers who work twenty times faster than anyone else. (I've got good people, but they aren't mythical.) My teams simply get more value out of every hour.

You could experiment with all the methods I've tried, but that's time-consuming and tedious. Though my teams and I have learned from every experiment, not all have been of equal value. My current and former teams still use design patterns, unit testing, continuous integration, design and code reviews, static analysis, planning poker, pair programming, test-driven development, and continuous deployment to varying degrees. However, it was Scrum that had the biggest impact on team productivity and quality—that is, until we switched to Kanban four years ago. With Kanban, for the first time in my long career, I can honestly say that every minute of work my teams do adds value for customers to our products. No time or effort is wasted, and quality is assured.

This book is about how you can duplicate my success with your teams. I've done all the experimenting. I've taken all the missteps. I've culled what's important, and I've laid it out for you in plain language and straightforward steps so that you get just the benefits. Don't let your peers read this book, make use of Kanban, and start making you look antiquated. Take the easy steps I describe and start producing better software that customers love—with high quality, on time, and at low cost.

Who should read this book

This book is for practicing or aspiring software development professionals. You might have started creating software in the 1960s or are just now graduating from college. You could be part of an established software company or an IT organization within a larger company, or you could be a do-it-yourself app or web developer. You might be a software analyst, project manager, program manager, developer, tester, project lead, or development manager. So long as you are a serious practitioner of software development, you will find this book enlightening and invaluable.

This book provides pragmatic and prescriptive step-by-step instructions on how to produce the most value for your customers, with the highest quality at the lowest cost in the least amount of time. I've included diagrams, tables, charts, worksheets, rude Q & A sections, and troubleshooting sections to clarify concepts and guide you toward success. This book also has chapters especially for people who want to adapt from traditional Waterfall methods or evolve from Scrum.

This book might not be for you if . . .

Although the last chapter, "Further resources and beyond," covers the basic theory behind Kanban and other techniques, this book might not be for you if you're looking for a deep reference. Students, academics, and consultants might prefer a different

text for in-depth analysis of theory and practice. I suggest several such texts in the last chapter.

Organization of this book

The book follows the progression that a feature team (3–10 people) might experience when learning Kanban:

- Chapter 1, "Getting management consent," covers approaches and steps for gaining consent from management to use Kanban (a necessary condition before you start). This chapter includes an open letter to your manager with a sample proposal.
- Chapter 2, "Kanban quick-start guide," can get you going with Kanban within a few days, provided you have an existing backlog of work. The chapter also includes a troubleshooting section.
- Chapter 3, "Hitting deadlines," helps team members fill and order their backlog as well as estimate how long their project will take and how many resources they'll need.
- Chapter 4, "Adapting from Waterfall," and Chapter 5, "Evolving from Scrum," are for teams that currently use traditional Waterfall or Scrum. These chapters summarize the argument for using Kanban, provide the steps to adapt or evolve to Kanban, and answer the questions a team might have. These chapters and their rude Q & A sections are based on my direct experience with introducing traditional Waterfall and Scrum teams to Kanban.
- Chapter 6, "Deploying components, apps, and services," focuses on delivering the value you produce with Kanban to customers—everything from continuous integration to continuous deployment.
- Chapter 7, "Using Kanban within large organizations," is for teams that use Kanban within large projects of hundreds or thousands of engineers, including how to fit in and report up.
- Chapter 8, "Sustained engineering," is a special contribution from James Waletzky about how to apply Kanban to perform postrelease software maintenance.
- Chapter 9, "Further resources and beyond," provides an overview of the theoretical underpinnings of Kanban and covers how you can improve beyond

the practices described in the previous chapters. This chapter provides resources for those who want to continue learning and evolving.

Acknowledgments

I'll start by congratulating Microsoft Press on its thirtieth anniversary. It was at the anniversary party in Redmond that Devon Musgrave approached me about writing this book. Many thanks to Devon for his suggestion and for championing the book's publication. I'm also deeply indebted to my editor, John Pierce, who did wonders to the readability and consistency of my words.

This book had six reviewers: David Anderson, Corey Ladas, Richard Hundhausen, James Waletzky, Doug Thompson, and Vijay Garg. Doug and Vijay currently work on two of my teams and use Kanban every day. Their feedback was essential to the clarity and accuracy of the text and its examples. James Waletzky is a passionate practitioner of Agile. We've worked together in the past, and his candor and critique have guided my writing for years. James, in his awesomeness, also supplied the chapter on sustained engineering (Chapter 8). Rich joined this project late but provided tremendous suggestions and a tight connection back to the Agile Project Management series. In all, I believe you can't produce worthwhile designs, code, or commentary without thoughtful expert review. To the extent that this book is worthwhile, it is due to my exceptional reviewers.

I want to especially recognize David Anderson and Corey Ladas. David has been an industry leader in project-management techniques throughout his career. He is the originator of the Kanban Method for evolutionary improvement. David speaks and trains professionals around the world. David has always been generous with his time and insights ever since we first collaborated at Microsoft years ago. David's contributions to this book's accuracy, framing, and language are essential and extensive. Even with all his traveling, David found substantial time to carefully review this work, for which I am immensely grateful.

Corey Ladas's influence on my thinking and career cannot be overstated. Corey introduced me to Scrum, Agile, axiomatic design, TRIZ, House of Quality, and numerous other techniques. In 2007, Corey invited me to Corbis to see the work that he and David Anderson were doing there with Kanban. I was instantly enthralled. Although it would take me a few years to try it myself, I immediately shared the work with as many peers as would listen. Corey is a deep thinker, who consistently challenges the status quo. He is fearless and unflinching. Corey can be tough and defiant, but he is always honest and insightful. I am delighted to call him my friend. Corey was the inspiration for this book.

Finally, I'd like to thank my past managers (Curt Steeb, Boyd Multerer, and Kareem Choudhry) for making Xbox such a great place to work, all my team members over the years who embraced experimentation and shared in my success, and, most of all, my wife, Karen, and my sons, Alex and Peter, for making me so very happy.

Downloads: Sample files

I've provided a couple of sample files, which you can download from the following page:

http://aka.ms/pmwithkanban/files

The first file is a sample of a letter and proposal you can provide to your management to gain consent to use Kanban. The second is an Excel workbook with every sample spreadsheet shown in the book, including those to calculate work-in-progress (WIP) limits, estimate completion dates, and even chart productivity and quality over time.

Follow the instructions on the page to download the files.

System requirements

The files provided online are in the Office Open XML format. The basic requirement for using the files is to have an Excel Viewer and Word Viewer installed on your computer.

You can download the Excel viewer from *http://www.microsoft.com/en-us/download/details.aspx?id=10*.

You can download the Word view from *http://www.microsoft.com/en-us/download/details.aspx?id=4*.

Errata, updates, & book support

We've made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

http://aka.ms/pmwithkanban/errata

If you discover an error that is not already listed, please submit it to us at the same page.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to *http://support.microsoft.com*.

Free ebooks from Microsoft Press

From technical overviews to in-depth information on special topics, the free ebooks from Microsoft Press cover a wide range of topics. These ebooks are available in PDF, EPUB, and Mobi for Kindle formats, ready for you to download at:

http://aka.ms/mspressfree

Check back often to see what is new!

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://aka.ms/tellpress

We know you're busy, so we've kept it short with just a few questions. Your answers go directly to the editors at Microsoft Press. (No personal information will be requested.) Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*

This page intentionally left blank

CHAPTER 4

Adapting from Waterfall

This chapter is for people currently using a traditional Waterfall method for product development. If your team uses Scrum, please feel free to skip to the next chapter, "Evolving from Scrum."

Kanban is simple in structure and uses common terminology. As a result, a wide range of people can begin using it without significant trouble or prolonged explanation. As with any new method, it takes a few weeks to adjust to Kanban and a few months to master it. Nonetheless, even if you learned product development decades ago, you can quickly get started and feel productive with Kanban. After a couple of months, the increases in productivity, quality, predictability, and agility should be evident and measurable to your leadership and your team.

When I talk about "traditional Waterfall," I mean the practice of writing specs, implementing features, and performing validation in bulk (many features at once) over the course of milestones that often span months. I've experienced many variations of Waterfall at Microsoft, Boeing, and other places where I've worked.

This chapter will help you adapt your variation of traditional Waterfall to Kanban without much fuss or hassle. I've even included a rude Q & A listing questions that a blunt team member might ask, followed by pragmatic answers meant to reassure, build trust in the new approach, and clearly explain how to achieve great results with Kanban.

The topics covered are:

Introducing Kanban to a Waterfall team
Working in feature teams
Completing features before starting new ones
Dealing with specs and bugs
Engaging with customers
Celebrating performance improvements
Rude Q & A
Checklist

Introducing Kanban to a Waterfall team

A traditional Waterfall team is one that likely has members who've been doing product development for decades. Their habits were formed long ago and have served them well over the years. Although the products they build might be buggy initially, the members of the traditional Waterfall team know

how to stabilize the product at the end, drive out the bugs, and release reasonably close to the scheduled date with acceptable quality.

However, as product development has moved to shorter timelines, and long stabilization periods are no longer viable, traditional Waterfall teams may be pressured to become "agile." This can make team members feel both uncomfortable with the notion of change and disrespected given their past accomplishments.

For a traditional Waterfall team to embrace Kanban, you need to explain why a change is necessary, utilize people's valuable Waterfall experience, and lightly adjust their familiar methods to facilitate a smooth workflow and quick cadence. Once the team masters Kanban, it can choose to improve further by making more significant adjustments to its approach. However, the starting point can feel familiar and straightforward.

To understand why making some adjustments to Waterfall is necessary, it's helpful to recognize where a traditional Waterfall approach breaks down:

- When traditional Waterfall is done well, you start with a solid plan everyone believes in (based on market research, prototyping, architectural design, and other planning), with clearly documented requirements and specifications and a thoughtful schedule based on known dependencies and staff allocations. Such plans typically prescribe a year or more of product development, broken into a series of milestones followed by a prolonged stabilization period.
- Unfortunately, within a few months of implementing the plan, uncertainties creep in, requirements change, dependencies shift, and people move around. As a result, the team has to update plans, specifications, and schedules and throw out or rework the portions of the product (and associated tests, if any) that are affected.
- The cycle of updates, discarded work, and rework repeats as the market shifts during product development and customers provide feedback at each milestone.
- All the plan and product churn results in numerous quality issues that often extend the already long stabilization period, frequently delaying the product release.

Ideally, every product change should receive immediate customer feedback, adjusting to market shifts daily, with no buildup of churn or other quality issues. Doing so would require a smooth flow of work through each development step and a continuous development approach that delivers completed, high-quality product improvements every day.

Tip Even secret products and new breakthrough products have customers and benefit from frequent customer feedback. You might need to be selective in choosing the customers you use, but their feedback is essential for delivering a high-quality, delightful product.

Scrum tries to address the breakdowns in traditional Waterfall by converting the milestones to short sprints (typically one to four weeks in length), producing customer-ready product

improvements each sprint, and adjusting to customer feedback at the end of each sprint. It's a substantial improvement, but churn still builds up during sprints, plans still need to be updated each sprint, and the flow of customer-ready product enhancements, with its immediate customer feedback, is more frequent but still not smooth or continuous.

Kanban is built for smooth and continuous delivery of customer value. It carefully controls the flow and quality of work to discover and resolve issues immediately. It limits the work in progress (what's called *inventory* in manufacturing) so that churn doesn't build up and the team and product can adjust to market shifts daily. Kanban does all this while working within the current roles and specialties of traditional Waterfall teams, making it seem familiar and straightforward.

When introducing Kanban to a Waterfall team, start by reassuring team members. Tell them, "Nearly everything you did before to develop products, you still get to do. You don't have to learn new roles or unfamiliar terminology. We will put a big focus on quality, and we will do our best to frontload the most critical work. But you will still do day-to-day product development the way you always have. The biggest change will be the way you select what to do next. Luckily, that selection process is easy and displayed on a big board where everyone can see it."

At this point, you can proceed to the Kanban quick-start guide (Chapter 2) using your current backlog of work as a starting point. But before you move on, here are a few points about the rest of this chapter.

- A couple of areas that might take traditional Waterfall team members by surprise are worth discussing in advance. I cover these in the next two sections: "Working in feature teams" and "Completing features before starting new ones."
- After your feature team has gotten used to Kanban, you might choose to change how you deal with specs and bugs and how you engage with customers. I describe those changes in the sections "Dealing with specs and bugs" and "Engaging with customers."
- To show your management and your team how Kanban is improving productivity and quality, you'll want to measure and celebrate your progress. This can be critical to gain commitment to Kanban, it's easy, and it provides a nice morale boost. See the "Celebrating performance improvements" section.
- Finally, I have answers to common questions in the last section, "Rude Q & A."

Inside Xbox

I've moved three traditional Waterfall teams to Kanban: two software development teams and one operations team. A few people on the teams were familiar with Scrum and agile techniques, and some had worked with virtual feature teams, but most had only used traditional Waterfall.

Complaints were rare during adoption. Team members found Kanban straightforward and the signboard quite helpful. Most of the confusion and questions were about the WIP limits and dealing with occasional work items that didn't match typical work.

I observed the first few standup meetings with each team and answered any questions that arose. Each team used the Specify, Implement, and Validate steps I describe in the Kanban quick-start guide. Those steps are familiar to traditional Waterfall teams. (Some of my teams called the Specify step "Breakdown" but still used it for specification.)

After the first few standup meetings, I attended only occasionally. When questions came up about WIP limits or unusual work items, team members would stop by my office and ask, "What are we supposed to do?" I captured those questions, and the answers, in the troubleshooting section of the Kanban quick-start guide in Chapter 2.

Working in feature teams

Kanban brings feature teams together each day to view the signboard and handle blocking issues. A feature team is a group of individuals, often from multiple disciplines, who work on the same set of product features together.

A typical feature team might have 1–3 analysts, 1–6 developers, and 1–6 testers (a total of 3–15 people), but some can be larger. Feature teams may also have marketers, product planners, designers, user researchers, architects, technical researchers, data scientists, quality assurance personnel, service engineers, service operations staff, and project managers. Often, feature team members are part of multiple feature teams, although developers and testers tend to be dedicated to a single team.

Many people who use traditional Waterfall work on feature teams, all for the same manager or as a virtual team. However, some groups completely separate different disciplines, using formal handoff procedures between disciplines, including associated documentation.

With Kanban, you can maintain separate disciplines and formal handoff procedures if you prefer. The handoff procedures map directly to the done rules for each step. However, Kanban does require each discipline working on the same workflow to share the same signboard and attend standup together. While this is certainly a change, it's a relatively minor logistical one that is easily incorporated into people's workday.

The key is to pick a time for the standup when all feature team members can attend. My teams schedule theirs at 10:30 a.m. It's late enough in the morning that even folks who sleep in arrive on time, and it's early enough that no one is away at lunch or at an afternoon obligation. Since standup takes only 5–15 minutes, even with large teams, it's over before 10:45 a.m. For teams with remote members, pick the best time you can and use online meeting tools.

While the focus of the standup meeting is to look over the signboard and handle blocking issues, getting everyone together also opens opportunities for cross-discipline team members to connect. Keep the standup focused properly, but after the standup, people can meet and sync up on a variety of scheduling, process, and design issues, while folks not involved in these issues return to their

work. It's invaluable to have this regular time when everyone working on the same features can align themselves.

To help everyone adjust to the daily standups, tell your team, "We all work together to create great features for our customers. When we work together, it's helpful to get together daily, see our work progress, and handle any issues that might keep our high-quality work from reaching our customers quickly."

Completing features before starting new ones

Some traditional Waterfall teams specify (and review) every feature in a release before any features are implemented, and some implement every feature before any are validated. Some Waterfall teams arrange release work into a series of Specify/Implement/Validate milestones. Within each milestone, all features for that milestone are specified before they are implemented and implemented before they are validated. This traditional Waterfall approach is reminiscent of batch processing jobs with mainframe computers.

The batch approach in traditional Waterfall is simple and keeps team members focused. However, it can be inefficient and inflexible if the batches are large and can't be easily changed or reordered. Arguably, the biggest change for traditional Waterfall folks adapting to Kanban is to work on very small batches of items, constrained by WIP limits. The active cards in a step form the current batch. Instead of milestones or releases lasting months or even years, the batches last only days at a time. (I talk about coordinating Kanban work within much larger projects in Chapter 7, "Using Kanban within large organizations.")

Although the small batches in Kanban are a new concept, traditional Waterfall folks adapt quickly because the steps they follow to process each batch are the same as they were in traditional Waterfall. They just process only a few items at a time. Typically, two areas of confusion or surprise come with the shift to small batches:

- There's initial confusion about what to do when the WIP limits restrict the flow of batches. I cover those cases in the troubleshooting section of the Kanban quick-start guide (Chapter 2).
- There's confusion or surprise about the increased frequency of cross-discipline interaction. Because Kanban uses very small batches, people responsible for different steps engage with one another more often. At first, this can seem like a distraction. Personal expectations of quiet time, appropriate engagement, and other professional courtesies develop quickly. In time, team members view the increased interaction as a significant improvement because it catches design and implementation flaws early, before they permeate the entire release and become more expensive to repair.

To smooth the adaptation to small batches, tell your team, "We're all familiar with specifying features, implementing them, and validating that they work well for customers. We used to specify, implement, and validate a whole bunch of features at once. It was easy to organize around large batches, but the world changes quickly, and turning big boats takes time. We're now adopting small

batches, moving a card on our signboard as each feature, each bit of customer value, gets specified, implemented, and validated. The cards make tracking these small batches easy, and the small batches make it easier to keep pace with the world and the needs of our customers."

Once your feature team is used to meeting together for the daily standup, and the team has become familiar with the dynamics and extra interaction of working on small batches, you might want to introduce and make a couple of more adjustments to further improve your smooth and continuous delivery of customer value. You might want to change how you deal with specs and bugs and how you engage with customers. I describe those changes in the next two sections.

Dealing with specs and bugs

When you complete features before you start new ones, you significantly shorten the time between writing specs and implementing them and between creating bugs and fixing them. In practice, that reduced time leads to the simplified handling of specs and bugs in Kanban.

Specs

Many traditional Waterfall teams write detailed design specification documents ("specs") for every feature, all of which are reviewed and approved before implementation starts. Detailed specs are important because in traditional Waterfall a feature may not be implemented until months after specification, and not be validated until months after implementation. If you don't clearly document the feature in detail, developers won't remember what to implement and testers won't remember what to validate.

Because Kanban uses small batches, the time between specification, implementation, and validation is measured in days instead of months. At first, traditional Waterfall teams might choose to continue writing detailed specs for every feature. However, teams may find that level of formality to be unnecessary because the design is still fresh in everyone's minds.

It's still important to document, review, and approve designs. However, it's often not necessary to transcribe a design and a list of considerations drawn on a whiteboard into a formal document, with detailed explanations of every point so that people remember them months later. Instead, teams often use electronic notebooks, wikis, or other quick authoring tools to capture photos of whiteboards and key points of design discussions. (My teams love OneNote for this purpose.) Those informal documents capture enough of the design for everyone to remember the main attributes, issues, and agreements. Then, as the feature is implemented, details of the design emerge and can be discussed in real time instead of being speculated upon months in advance.

As I point out in the "Troubleshooting" section of the Kanban quick-start guide (Chapter 2), some feature areas or individual features are still so complex that a detailed design document is necessary. Choosing to write a detailed spec should not be a matter of dogma or habit. It should be a decision based on the needs of the team (and the customer, should the customer require it). If a feature or feature area is unclear, or the tradeoffs and architecture are in question, you should write a detailed

spec and flesh out the design. Otherwise, a quick, informal electronic notebook or wiki should be sufficient. (My teams use Word for detailed specs and OneNote for other documentation.)

Bugs

In traditional Waterfall, features might not be validated until months after implementation. On a large project with hundreds of engineers, validation may find thousands of bugs. Often, validation takes as long as or longer than implementation.

Over the years, traditional Waterfall teams have devised a variety of ways to handle large bug counts:

- With limited time to fix so many bugs, each bug must be prioritized and duplicate bug reports removed. At Microsoft, we call this process "bug triage." Team leaders representing each job role meet for roughly an hour each day and review every new or updated active bug. They discuss the impact of the bug (severity, frequency, and percentage of customers affected) and set an appropriate priority (fix now, fix before release, fix if time, or fix in subsequent release). They can also decide that the bug is a duplicate of a prior reported issue or that the bug isn't worth fixing (usually because a trivial workaround is available, most customers won't encounter the bug, or the fix would cause more havoc than the bug).
- Some teams have "bug jail," in which individual engineers must stop further development and only resolve bugs until their individual bug counts drop below a reasonable level (such as below five bugs each).
- Some teams have something like "workaholic Wednesdays"—one day a week when the team doesn't go home until all bugs are resolved, or at least brought below a reasonable level (such as below five bugs per engineer).
- Every traditional Waterfall team I've encountered has substantial stabilization periods at the end of each milestone or at the end of each release. During stabilization, the team (and often the entire organization) focuses on nothing but fixing bugs, doing various forms of system validation, and logging any new or reoccurring bugs they find. Stabilization for a large project can sometimes last longer than all the specification, implementation, and prestabilization validation times put together.
- Some progressive development teams might employ extensive code reviews, inspections, unit testing, static analysis, pair programming, and even test-driven development (TDD) during implementation to reduce the number of bugs found during validation. These methods make a big difference, but you are usually still left with a substantial number of bugs at the end, partially because system validation happens well after implementation, and partially because adherence to these practices varies widely among developers.

In Kanban, a bug's life is a bit different:

- Kanban's small batches ensure that validation happens only days after implementation, so bug backlogs are small and fixes are readily apparent.

- Kanban ensures that every task implemented has been through code review, inspected, unit tested, statically analyzed, pair programmed, or designed and verified using TDD, based on the implementation done rule that the team imposed on itself. Even reckless developers are kept in line by their own teams. (No one likes cleaning up after a lazy slob.) This further reduces bug counts.
- Likewise, the validation done rule ensures that every task has gone through integration testing and all its issues are resolved. Thus, by the time a task is done with validation, it's ready for production use. I talk about taking advantage of this fact in Chapter 6, "Deploying components, apps, and services."

Even though every work item completing validation each day has gone through integration testing and all its issues are resolved, bugs can still be discovered in production use, stress testing, security testing, usability and beta testing, and a variety of other product-wide system testing. However, the stabilization period required to fix those issues is far shorter, and in my experience, the number of stabilization bugs opened per feature team drops from hundreds to 10 to 20 (a couple of weeks of effort to resolve).

After adapting to Kanban, traditional Waterfall teams might choose to continue bug triage, bug jail, workaholic Wednesdays, and long stabilization periods. However, teams may soon find some or all of those practices unnecessary. In a large organization, a cross-team, product-wide triage stabilization period might still be needed (see Chapter 7), but individual teams using Kanban won't have enough bugs to make team triage, bug jail, or workaholic Wednesdays useful.

Engaging with customers

Traditional Waterfall teams typically engage with customers during planning (before work on the release begins), every one to six months at the end of stabilization for each milestone (perhaps as part of a preview program), and during final release stabilization (often as part of a beta program).

As experienced engineers know, customer feedback is invaluable for specifying, implementing, and validating product improvements. Customer engagement with traditional Waterfall teams is limited because the product is usually too buggy or incomplete to use. Customers can't provide actionable feedback when they can't use the product. Instead, customers must wait until the end of stabilization periods, when the product has the fewest-known bugs.

In contrast, Kanban provides an opportunity to engage with customers at whatever cadence the team or its customers finds most convenient, including continuously with customers who are onsite. Kanban enables this because when a task is through validation, it's ready for production use.

Naturally, you want to try out product improvements with a limited number of customers first, and gauge their reactions, before publishing those improvements to your entire customer base. One common approach is to first have the entire product development team try the latest version, then share it through an early adopter program, then publish it to a broader preview audience, and then finally release it to all your customers. (I describe an example in the following "Inside Xbox" section.)

The customer feedback you get along the way can be used to hold back changes that don't work, adjust designs, reorder pending work, find subtle or rare bugs, fill gaps in usability scenarios, and expand features in areas that customers love. Since Kanban limits work in progress (small batches), it's easy to adjust to customer input within days.

To take advantage of the customer feedback opportunity, you need to establish a preview program, an early adopter program, and a way for the entire product development team to try the latest version. Here are some possibilities:

- Many traditional Waterfall teams have beta programs already. You can repurpose your beta program as a preview program.
- To establish an early adopter program, you can engage the most active members of your beta audience or hand-select key customers and offer them early adopter status as a perk.
- To expose the entire product team to the latest builds, follow the steps I outline in Chapter 6.

Soon, you'll be enjoying greater confidence, higher quality and usability, and more delighted customers. There's nothing quite like engaged customers providing actionable feedback on working products.

The last, and arguably most critical, step to ensure that your team and your management commit to Kanban is to measure the great results you're getting, and then celebrate your success. That's the topic of the next section.

Inside Xbox

Xbox One launched in the fall of 2013, and at the time of this writing has shipped new releases of the platform every month since. The Xbox One product team uses a variety of methodologies: Scrum, Scrummerfall (traditional Waterfall with short, fixed-length milestones called "sprints"), variants of Extreme Programming (XP), and Kanban. All of these techniques enable frequent customer feedback.

The move to monthly releases had its challenges, but was surprisingly smooth and natural. In hindsight, I feel that two key changes made this smooth transition possible:

- Roughly 18 months in advance, we synchronized all enterprise Scrum and Scrummerfall sprints and XP iterations to match a shared 4-week cadence. Kanban releases continuously and can match any cadence, so the Kanban teams didn't need to adjust.
- In the months leading up to the launch, we released new platform builds to the Xbox product team weekly, then to the early adopter program the week after, and then to a preview audience monthly. We released fixes immediately for issues found by each audience. Once Xbox One launched, we simply continued that release cadence and approach, adding a broad release to all customers a few weeks after the preview audience received it.
- Establishing the various release audiences was pretty easy, relatively speaking.

- We already had a release mechanism for the weekly Xbox product team builds. It's called "dog food" (as in "eating our own dog food"), and has been around since the first version of the Xbox. Naturally, we needed to update the program for Xbox One, but that was anticipated.
- We already had a passionate early adopter program. Basically, we ask all full-time Microsoft employees if they're interested in early releases of Xbox—first come, first served. The list fills within hours. Again, we needed to update our deployment process for Xbox One, but that was expected.
- And we already had a passionate Xbox beta audience that we could convert over to Xbox One preview. This was the most work—not to get the audience, but to reconfigure the beta feedback site, logistics, and tooling to a monthly release cadence.
- Now that Xbox is releasing once a month, we never want to go back. Our customers love the new features and responsiveness to requests. Our engineers love the predictability, the decreased churn between releases, and the reduced pressure (if your feature misses one month, it can always go out the next). And our management loves the positive press we've received and how that's translated to the bottom line.

Celebrating performance improvements

Even if your traditional Waterfall team takes easily to Kanban and likes the change, members may still question whether the change was worth it. After all, traditional Waterfall does work, and has likely worked for years. Your management probably has a similar concern: Was the effort to adopt Kanban worth the cost?

While each team has its own reasons, you likely adopted Kanban to increase agility and deliver more high-quality value to your customers in less time. The good news is that it's straightforward to measure those outcomes. By measuring them from the start, and showing your team and management how your results improve over time, you'll enhance everyone's commitment to the change and boost morale and team pride in the process.

I'll focus on two daily measurements and their moving averages to capture agility, productivity, and quality. I selected these particular measures—completed tasks and unresolved bugs—because they are easy to calculate for both Waterfall and Kanban, they have clear definitions, and they relate directly to agility, productivity, and quality. Here's a breakdown of each measure:

	Completed tasks	Unresolved bugs
Definition	Count of the team's tasks that completed validation that day with all issues resolved	Count of the team's unresolved bugs that day
Concept measured	Day-to-day agility	Day-to-day bug debt
Moving average captures	Productivity	Product quality
Waterfall calculation	Count the rough number of the team's tasks in each spec that completed validation and had all issues resolved by the end of a milestone or release, and associate that count with the end date of the respective milestone or release.	Extract the number of the team's unresolved bugs by day from your bug-tracking system.
Kanban calculation	Count the number of the team's tasks that completed the done rules for validation on each date.	Extract the number of the team's unresolved bugs by day from your bug-tracking system.
Caveats	Only measures productivity of task completion, not other daily activities or tasks without specifications.	Only measures aspects of product quality captured in bugs, but doesn't account for different types of bugs.

Notes on these measures:

- If the size of your team varies dramatically, you should divide by the number of team members, resulting in completed tasks per team member and unresolved bugs per team member. Doing so makes the measures easier to compare across teams but less intuitive to management, in my experience.
- The completed tasks calculation for Waterfall bulks all the tasks at the end of each milestone or release because that's the earliest date when the tasks are known to be validated with all issues resolved. However, if your Waterfall team associates all bugs directly with tasks, you can count tasks toward the day when their last bug was resolved. Getting this extra accuracy is nice, but it isn't essential. In particular, the extra accuracy doesn't matter for the moving average (the productivity measure), so long as you average over the length of your milestones.
- The moving average can be a 7-day average for measuring weekly productivity, a 30-day average for monthly productivity, or whatever length you want. To compare the productivity of Waterfall to Kanban, you should average over the length of your Waterfall milestones.

Figure 4-1 and Figure 4-2 are examples of plots of completed tasks and unresolved bugs over four nine-week periods (included in a downloadable worksheet; see the book's introduction for more details). The first two periods are Waterfall milestones, split into six weeks of implementation and three weeks of validation. The second two periods are Kanban running continuously. The data is based on the team configuration I used as an example in Chapter 3, "Hitting deadlines." I have the team working seven days a week (for simplicity, not punishment). Their pace of task work remains the same throughout all four periods, which is artificial but unbiased. (I do add a little random noise throughout to make the results a bit more realistic.)

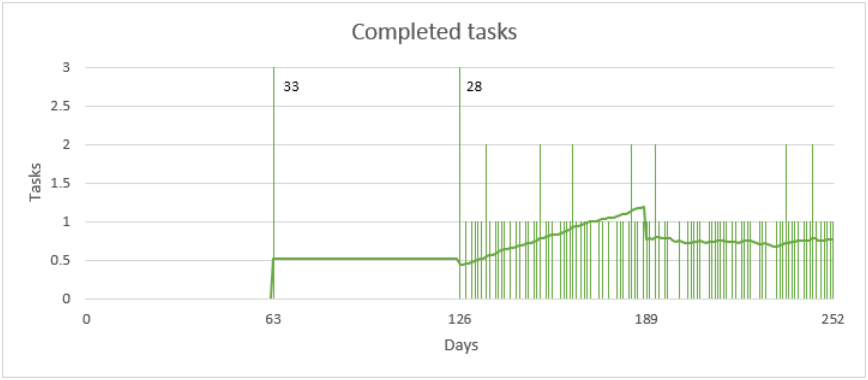

FIGURE 4-1 Plot of completed tasks over four nine-week periods, with a superimposed running average.

Looking at the chart of completed tasks, all tasks are considered completed on the final day of validation during the first two Waterfall periods. The two columns of completed tasks from those periods go about 10 times the height of the chart. I've zoomed in to better view the nine-week moving average line measuring productivity. It starts at around 0.52 tasks per day (33 tasks / 63 days), increases to around 1.2 tasks per day as the results from the second Waterfall milestone combine with the Kanban results, and then settles to around 0.76 tasks per day as the steady stream of Kanban work establishes itself.

The improvement from 0.52 to 0.76 represents a 46 percent increase in productivity. That's the equivalent to working five days and getting seven and a half days of tasks completed—it's like you worked hard through the weekend and still got the weekend off. The day-to-day agility of Kanban is also immediately apparent, as work is completed daily instead of being delivered in bulk.

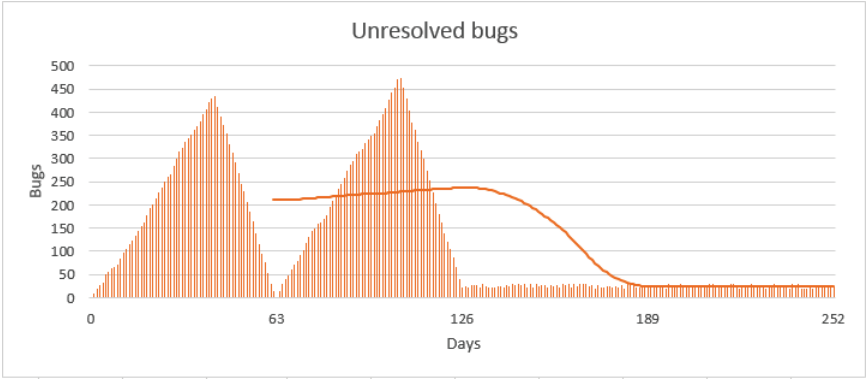

FIGURE 4-2 Plot of unresolved bugs over four nine-week periods, with a superimposed running average.

The unresolved bugs chart in Figure 4-2 displays a fairly typical Waterfall pattern of building up a bug backlog during the six-week implementation phase of the milestone and then resolving those bugs during the three-week validation phase. This shows as the two inverted V shapes, one for each

Waterfall milestone. This simple example has a fairly uniform bug resolution rate during validation. In practice, bug resolution rates often fluctuate significantly as complicated issues are uncovered and some fixes regress old problems.

Once the Kanban period starts, there is a fairly steady stream of unresolved bugs, but it never builds up. Bugs are kept in check by the WIP limits and done rules. As a result, the nine-week moving average line measuring quality improves dramatically with Kanban. The product is always ready for production use and customer feedback.

Seeing these dramatic improvements in productivity and quality should be enough to warm the hearts of hardened cynics and skeptical managers. Sharing them weekly with your team and your management gives you all something to celebrate (in addition to the increased customer value you deliver).

What's more, with Kanban, the quality level is steady throughout the product cycle, so the prolonged stabilization periods associated with traditional Waterfall are gone or, in large organizations, reduced to only two or three weeks of system-wide testing. Replacing months of stabilization with a few weeks at most yields more time for continuous product enhancements, making the measurable improvements in productivity and quality even more remarkable.

However, a month or two will pass before you begin to see great results. The cynics and skeptics on your team are bound to have questions. It's time for the rude Q & A.

Rude Q & A

What follows is a rude Q & A session that is based on questions I've received from traditional Waterfall team members when they were introduced to Kanban. I hope it covers many of the questions you receive.

Q Why are we adopting Kanban?

A The software industry continues to evolve. Customers now expect their apps and services to update automatically and be improved continuously. Our current practices impede us from keeping pace.

Q Instead, how about we stop changing practices, plans, and requirements constantly?

A Plans and requirements always change, no matter how well prepared or well considered they are. Even when you are your own customer, you constantly change your mind. Nevertheless, stable plans, requirements, and practices aren't sufficient for us to react quickly to the current market.

Q So you're saying it's hopeless?

A It's not hopeless at all. The rest of the industry is moving forward. We can join in with some relatively small changes that have a big impact.

Q Will we need training for Kanban?

A We'll learn Kanban mostly from doing it every day. Books about Kanban and Kanban coaches are available to help us with adoption.

Q Are you going to make me sit in a cubical farm or share a desk?

A No, you don't have to change offices or share desks. You don't even have to change how you write specs, software, or tests. We're just adding a daily standup meeting at which we track our workflow on a large board.

Q How is adding another meeting going to make me faster?

A The daily standup lasts less than 15 minutes and replaces our old planning and status meetings. We'll schedule it at a convenient time for the whole team. Regardless, it's smooth and continuous workflow that makes the team faster.

Q How do we get smooth and continuous workflow?

A Today, we specify, implement, and validate features in bulk. It's fast, but when a plan or requirement changes, we lose work and momentum, and when bugs are found late, we have lots of costly rework. Going forward, we'll pace specification, implementation, and validation to run at similar rates so that we can smoothly complete small batches continuously.

Q How do small batches make us faster?

A Small batches speed us up in a variety of ways:

- When a plan or requirement changes, it affects only a small batch, so less work is wasted and the team can adjust quickly.
- Small batches mean smaller bug backlogs, so we don't need dedicated team stabilization periods.
- We have to specify, implement, and validate only a small batch at a time, so we don't write specs that aren't implemented, implement code that isn't validated, or write tests that are never run.
- We can get customer feedback after every small batch, which avoids being blindsided later and reworking major portions of the design (or releasing a product that customers dislike).

In all, small batches significantly reduce wasted effort and rework, while improving the quality of our product.

Q So why haven't we been using small batches all along?

A Small batches require a coordinated flow of work. If analysts specify more features in a week than developers can implement, specifications start piling up, and you're back to doing work in bulk. Likewise for developers implementing more features than testers can validate. If you can't pace specification, implementation, and validation to run at similar rates, you're stuck working in bulk and dealing with wasted effort, large bug backlogs, and substantial rework.

Q We have a wide variety of analysts, developers, and testers. How do you pace work properly?

A You limit the pace of specification, implementation, and validation to match one another. You base those work limits on the current pace of each step, plus some buffer to account for the variety of people and work involved. These work limits are often called "work-in-progress (WIP) limits." Naturally, your initial guesses about the limits will need fine-tuning, but even initial guesses are typically good enough to produce a smoother, more continuous flow of small batches.

Q Wouldn't limiting work make us slower?

A We're not limiting people, we're limiting the kinds of work they do. If developers are stuck on a feature, an analyst specifying more features doesn't help. Instead, the analyst should work to unblock the developers (for example, clarify the spec, escalate an issue, research a customer preference, or bring in partner expertise). By focusing on keeping the work flowing smoothly, and limiting work so that it maintains a smooth pace, we actually get more work done in less time.

Q Why not keep the analyst working and have the developer move on to the next task?

A The key is the status of the blocked implementation task:

- If the task is blocked by an external dependency, the developer should move on to the next task until the old task is unblocked. We'll keep an eye on the blocked task in a special column on the signboard.
- If the task is blocked by a design question or issue, it's the analyst's responsibility to unblock the task, ideally while the issue is still fresh in the developer's mind. The limits on work and the visibility of blocked tasks prevent analysts from ignoring design issues and letting them fester.

The same applies for blocked specification and validation tasks. Instead of letting problems pile up, we're going to work together to fix them quickly. That keeps work flowing smoothly, fixes issues before they permeate the product and become harder to repair, and delivers value faster and more continuously to our customers.

Q Won't I be constantly hassled about fixing problems?

A Actually, you'll be hassled less than you are today. That's because we always have problems, but today we find and fix them late. By fixing problems early, they are fresh in your mind, easier to fix, and don't have time to multiply and cause further trouble. You will be hassled earlier than before, but most folks appreciate the dividends of not having problems fester.

Q Will everyone be coming to me with problems?

A No, you'll work on individual tasks like you always have. We're adding a daily standup meeting where we track our workflow on a large board. That signboard will show your work, and everyone else's work, on cards that flow across the three steps: Specify, Implement, and Validate. The WIP limits that set the right pacing will be written above each step on the signboard. Everyone will see when work is getting blocked and bunched up—they won't all just come to you. Instead, the team will work out issues together.

Q What if one person keeps causing all the blockage?

A There's a classic remedy if someone really can't do his or her job. However, if someone is careless or lazy and tends to pass on half-done work, he won't get away with it. We're going to have done rules for Specify, Implement, and Validate. Before someone can move a card, claiming it's done with the current step, the work has to pass the done rules for that step. No half-done work is allowed.

Q Who determines the done rules?

A You and the rest of the team decide on the done rules you'll follow. You know what causes issues. You know how to do your job well. You get to decide when a task should be considered done. Once the team decides on the rules, we'll write them at the bottom of the signboard so that there's no confusion. Anytime the rules need adjustment, we can change them together.

Q What else changes with Kanban?

A Nothing—that's it until we want to improve even further. For now, we'll have a daily standup in front of a big board with our work steps written on it, cards showing our work, limits on the number of cards at each step, and rules at the bottom of the signboard that determine when a card is done with its current step. Whenever you're done with a step, based on the done rules, you'll move your card on the signboard and grab the next available card. If no cards are available, that indicates that workflow is blocked, and you should work with your teammates to unblock the flow.

Q Where will the cards come from?

A We'll take our current feature list and work backlog, write the items on cards, bucket them in priority order, and place them on the left side of the signboard, in front of the specification step. When new work arrives or plans change, we'll add and rearrange cards as needed. (Details in Chapter 3, "Hitting deadlines," and Chapter 7, "Using Kanban within large organizations.")

Q When a card moves from one step to the next, who works on it?

A We'll assign work to whomever is free and capable of doing that work at the time. Pretty much like we do today, but as needed, not planned far in advance.

Q We have daily standups in front of a board that tracks progress. Are we already doing Kanban?

A Many traditional Waterfall teams meet daily in front of a board that's used to track progress. However, it's not Kanban unless you list your steps on the board (like specification, implementation, and verification), and each step has a work-in-progress (WIP) limit, a Done column, and a done rule clearly marked on the board.

Q Why does each step need its own Done column?

A Say you just completed implementing an item (it passes the implementation done rule). Without an implementation-specific check mark or implementation Done column, how would you indicate that the item is ready for validation? On an ordinary board, you just move the item to validation. However, that means the item is actively being validated and counts toward the validation WIP limit—neither of which is true. What's worse is that the item no longer counts toward the

implementation WIP limit, so you're free to implement another item, even if validation is overwhelmed and needs help. The Done column for each step clearly indicates the status of each item and controls the flow of items in conjunction with their WIP limits.

Q What happens to milestones with Kanban?

A The larger project might have release milestones to sync across teams, but we no longer need them as an individual team. Our small batches are always complete and ready for production use, based on our validation done rules. Learn more in Chapter 7, "Using Kanban within large organizations."

Q What about stabilization?

A The larger project might have release stabilization to resolve system-wide issues, but we no longer need it as an individual team. Our done rules ensure that work is done at the end of each step, with no remaining issues to stabilize.

Q If other project teams are using Waterfall, won't they still need stabilization?

A Yes, they will. While other teams stabilize, we can do a few different things:

- We can keep working on new tasks for the current project milestone. This might upset Waterfall folks or break project rules, so it may not be an option.
- We can work on new tasks for the next project milestone and check them in to a different source control branch.
- We can improve infrastructure and tooling and address other technical debt that has been neglected.
- We can train ourselves on new techniques and methods.
- We can determine the root cause of various issues we've encountered and seek to fix them.
- We can help other teams stabilize their code, particularly teams we depend on. This option may not be the most alluring, but it's the most helpful.
- We can run innovative experiments and acquire customer feedback.

Q What happens to planning?

A We'll still participate in overall release and project planning, but our team planning is simply a matter of ordering cards on our big board. We'll save a great deal of time and produce more value at higher quality, while easily and quickly adjusting to plan and requirement changes.

Q It sounds pretty simple. Aren't there sprints and burndowns or something?

A No, there's no unfamiliar terminology or new ways of doing the work. Kanban is pretty simple and direct. Kanban roughly means "signal card," "sign," "board," or "looking at the board." It refers to the cards that represent our work, the signboard displaying our workflow steps (and their assigned WIP limits and done rules), and our daily standup when we look at the signboard. There's nothing

more to it. We keep working on new cards within the WIP limits we've set, ensure that the cards are done with each step, and then work on the next cards. If cards start piling up, we figure out what's wrong and fix it so that work continues to flow smoothly and value is delivered continuously to our customers. It's a fast, simple, and easy way to work.

Checklist

Here's a checklist of actions for your team members to adapt from Waterfall:

- ❑ Explain why a change is necessary.
- ❑ Reassure team members that they can do their work as they did before, without learning new roles or unfamiliar terminology.
- ❑ Introduce daily standups that are attended by the entire feature team.
- ❑ Describe why it's important to work on features in small batches.
- ❑ Decide which features will require formal specification documents and which can be specified informally with whiteboard photos and notes.
- ❑ Determine how lower bug backlogs should affect bug-management and stabilization periods.
- ❑ Arrange opportunities for frequent customer feedback on the continuous value added to your products.
- ❑ Perform the actions listed in the checklist in Chapter 2, "Kanban quick-start guide."
- ❑ Measure completed tasks and unresolved bugs, or whatever productivity and quality metrics you choose, on a regular basis.
- ❑ Celebrate productivity and quality improvement (as well as continuous delivery of value to customers).
- ❑ Answer any questions team members have with respect and appreciation for their past accomplishments.

Index

A

acceptance test-driven development (ATDD), 121–122
Agile practice
  Kaizen and, 110
  mixing with Kanban, 120–123
Agile Project Management with Scrum (Schwaber), 57
Åhlström, Pär, 135
Amazon Web Services (AWS), 80
Anderson, David, 57, 118
Astels, Dave, 122
ATDD (acceptance test-driven development), 121–122
AWS (Amazon Web Services), 80

B

backlog
  adapting from Waterfall, 41, 45, 50, 52, 54
  checklist of actions to take, 24
  evolving from Scrum, 58–61, 65–68
  large organizations, 85–87
  ordering work, 28–29
  populating, 25–27
  SE workflow, 109
  troubleshooting problems, 17–23
Barry, Tanianne DeMaria, 120
BDD (behavior-driven development), 122
Beck, Kent, 121
behavior-driven development (BDD), 122
Benson, Jim, 120
Brooks, Frederick P., Jr., 34
buffer (drum-buffer-rope), 126–128
bugs, software
  adapting from Waterfall, 45–46, 48–51
  defined, 102
  evolving from Scrum, 62–65
  ordering work, 28–29
  sustained engineering, 109–111
  troubleshooting problems, 20
business applications, scaling Kanban, 118–119

C

chaos, setting limits on, 10–12
checklist of actions to take
  adapting from Waterfall, 56
  deadline management, 37
  deploying finished work, 83–84
  evolving from Scrum, 70
  Kanban, 24, 136
  large organizations, 100
  for management consent, 5
  sustained engineering, 115–116
Chelimsky, David, 122
completed tasks measurement
  adapting from Waterfall, 48–51
  evolving from Scrum, 62–65
conference presentations, 21
constraints
  defined, 11
  elevating, 125
  theory of constraints, 124–128
continuous delivery, 80
Continuous Delivery (Humble and Farley), 123
continuous deployment, 79–83
continuous integration, 72–75
continuous publishing, 77–79
continuous push, 75–77
core engineering team
  about, 102–103
  ownership, 104–105
  roles and responsibilities, 104
  troubleshooting problems, 112–113
Cox, Jeff, 126

Crenshaw, Dave, 124
critical chain, 129–130
CTE (current task estimate), 32–35
Cumulative Flow Diagram, 32–33
current task estimate (CTE), 32–35
customers and customer interaction
  adapting from Waterfall, 46–48
  communicating status to, 92–93
  demos for customers, 21
  evolving from Scrum, 67
  sustained engineering, 105–106, 111, 113

D

daily standup meetings
  about, 14–17
  adapting from Waterfall, 42–43, 52
  evolving from Scrum, 61, 68
  Personal Kanban and, 120
  sustained engineering, 115
  troubleshooting problems, 23, 115
De Bonte, Austina, 119
deadline management
  about, 25
  checklist of actions to take, 37
  establishing MVP, 27–28
  estimating features and tasks, 29–31
  ordering technical debt, 28–29
  ordering work, 28–29
  populating backlog, 25–27
  right-sizing teams, 33–37
  tracking expected completion date, 31–33
  Xbox example, 31
defects (Lean waste category), 131
deliver work step
  guidelines, 8–9
  troubleshooting problems, 21–23
demos for customers, 21
Dennis, Zach, 122
dependencies
  dealing with late or unstable, 94–98
  ordering work based on, 87–90
deploying finished work
  about, 71
  checklist of actions to take, 83–84
  continuous deployment, 79–83
  continuous integration, 72–75
  continuous publishing, 77–79
  continuous push, 75–77
  Xbox examples, 74–75, 77–79, 81–82
design work
  adapting from Waterfall, 44–45
  ordering, 28–29
  troubleshooting problems, 21, 23
DevOps practice, 122–123
distributed version control, 73, 75–77
distribution lists, communicating via, 93
done rule (pull criteria)
  about, 12–13, 15
  adapting from Waterfall, 54
  best practices, 13–14
  evolving from Scrum, 69
  implementing, 13
  specifying, 13
  validating, 13
drum-buffer-rope (TOC application), 126–128

E

escalations
  defined, 102
  in SE workflow, 109
  troubleshooting problems, 113
estimating features and tasks, 29–31
executive reviews, 21
expected completion date, tracking, 31–33

F

Farley, David, 123
features
  adapting from Waterfall, 42–44
  estimating, 29–31
Fletcher, Drew, 119
Fowler, Martin, 121
Freedom from Command and Control (Seddon), 135

G

global optimization, 132–135
Goal, The (Goldratt, et al.), 126
Goldratt, Eliyahu M., 126

H

Hellesoy, Aslak, 122
Helmkamp, Bryan, 122

high-level routine
capturing, 7–8
posting steps for, 8–10
hotfix, defined, 102
Humble, Jez, 123

I

implement work step
in daily standup meetings, 15–16
done rule, 13
guidelines, 8–9
troubleshooting problems, 17–20
improvement, product. *See* product improvement
incidents, defined, 102
introduction to proposal letter, 1–2
inventory (Lean waste category), 131

J

Jones, Daniel T., 135

K

kaizen, 110
Kanban
capturing high-level routine, 7–8
checklist of actions to take, 24, 136
defining done, 12–14
expanding to new areas of business and life, 117–120
improving beyond, 128–135
mixing Agile and Lean with, 120–123
redecorating wall, 8–10
running daily standup, 14–17
setting limit on chaos, 10–12
troubleshooting problems, 17–23
why it works, 123–128
Xbox switch to, 4–5
Kanban (Anderson), 118
Kessler, Robert, 122

L

Ladas, Corey, 57, 118
large organizations
checklist of actions to take, 100
communicating status up and out, 91–93
dealing with late or unstable dependencies, 93–97
deriving backlog from upfront planning, 86–87
fitting into milestones, 90–91
introducing Kanban within, 85
ordering work based on dependencies, 87–90
staying productive during stabilization, 98
Xbox examples, 89–90, 93, 97
Larrey, Dominique-Jean, 107
late dependencies, 94–95
lead time, 31
Lean-Agile Acceptance Test-Driven Development (Pugh), 122
Lean practice
global optimization and, 132–133
improving beyond Kanban, 129–132
Kaizen and, 110
mixing with Kanban, 120–123
Lean Software Development (Poppendieck and Poppendieck), 132
Lean Thinking (Womack and Jones), 135
LiquidPlanner, 87
Little's Law, 31, 123–124

M

management consent
checklist of actions to take, 5
importance of, 1
moving forward, 4–5
proposal letter to obtain, 1–4
Managing the Design Factory (Reinertsen), 124
McGrath and MacMillan Options Portfolio, 89–90
Microsoft Azure, 80
Microsoft Project, 87
milestones
adapting from Waterfall, 10, 39–40, 43, 49, 55
large organizations and, 85–86, 91, 95
minimalism, 123–124
minimum viable product (MVP), 27–28, 87
Modig, Niklas, 135
motion (Lean waste category), 131
MVP (minimum viable product), 27–28, 87
Myth of Multitasking, The (Crenshaw), 124
Mythical Man-Month, The (Brooks), 34

N

new work, troubleshooting problems, 21–22

North, Dan, 122

O

one-piece flow, 124
Options Portfolio (McGrath and MacMillan), 89–90
ordering work
  about, 28–29
  based on dependencies, 87–90
organizations, large. *See* large organizations
overprocessing (Lean waste category), 131
overproduction (Lean waste category), 130–131

P

pair programming, 122
Pair Programming Illuminated (Williams and Kessler), 122
performance improvements
  adapting from Waterfall, 48–51
  evolving from Scrum, 62–65
Personal Kanban, 120
Personal Kanban (Benson and Barry), 120
plan outline in proposal letter, 3
planning, deriving backlog from, 86–88
planning poker method, 30
PMs (program managers), 12
Poppendieck, Mary and Tom, 132
postrelease software maintenance. *See* sustained engineering
prioritization
  about, 28–29
  based on dependencies, 87–90
  MVP, 27
  SE triage teams, 107
  work items, 28
problem statement in proposal letter, 2
process details, troubleshooting problems, 23
product improvement
  estimating features and tasks, 29–31
  high-level routine for, 7–8
  sources of, 25–26
Product Owner (Scrum), 60–61, 67
program managers (PMs), 12
proposal letter
  introduction, 1–2
  plan outline in, 3
  problem statement in, 2
  risks and mitigation table in, 3
  solution statement in, 2
Pugh, Ken, 122
pull criteria (done rule)
  about, 12–13, 15
  adapting from Waterfall, 54
  best practices, 13–14
  evolving from Scrum, 69
  implementing, 13
  specifying, 13
  validating, 13

Q

Q & A session
  adapting from Waterfall, 51–56
  evolving from Scrum, 65–69
quick-solve meetings, 108
quick-start guide (Kanban)
  capturing high-level routine, 7–8
  checklist of actions to take, 24
  defining done, 12–14
  redecorating wall, 8–10
  running daily standup, 14–17
  setting limit on chaos, 10–12
  troubleshooting problems, 17–23

R

Refactoring (Fowler), 121
refactoring practice, 121
Reinertsen, Donald, 124
release managers, 12
requirement updates, troubleshooting problems, 21–22
reverse integrations, 72–74
risks and mitigation table in proposal letter, 3
root-cause analysis, 19, 110
rope (drum-buffer-rope), 126–128
RSpec Book, The (Chelimsky et al.), 122
rude Q & A session
  adapting from Waterfall, 51–56
  evolving from Scrum, 65–69

S

Scenario-Focused Engineering (De Bonte and Fletcher), 119
Scenario-Focused Engineering (SFE)
  design work and, 21

global optimization and, 132
scaling Kanban and, 118–119
Schwaber, Ken, 57
Scrum Master, 60, 67
Scrum method, evolving from
celebrating performance improvements, 62–65
checklist of actions to take, 70
introducing Kanban, 57–59
limiting chaos in, 10
mapping roles and terms, 60–61
replacing events, 61–62
rude Q & A session, 65–69
Waterfall method comparison, 40–41
Xbox examples, 59
Scrumban (Ladas), 57, 118
SE (sustained engineering). *See* sustained engineering
Sedden, John, 135
service packs, 102
SFE (Scenario-Focused Engineering)
design work and, 21
global optimization and, 132
scaling Kanban and, 118–119
signboards
about, 8–9
adapting from Waterfall, 42, 53–55
communicating status up and out, 92–93
continuous deployment, 81
continuous integration, 73–74
continuous publishing, 78
continuous push, 76
in daily standup meetings, 14–17
evolving from Scrum, 60–62, 67–69
ordering work, 28–29
positioning teams by, 9
Scenario-Focused Engineering, 118–119
sustained engineering, 109
troubleshooting problems, 113
Xbox example, 10
single-piece flow, 124
Six Boxes, 110
small batches. *See* WIP (work in progress)
software bugs. *See* bugs, software
software maintenance, postrelease. *See* sustained engineering
solution statement in proposal letter, 2
specify work step
in daily standup meetings, 16
done rule, 13
guidelines, 8–9
troubleshooting problems, 19, 21
Sprint Planning event (Scrum), 61–62, 66–67
Sprint Retrospective event (Scrum), 61–62, 66
stabilization periods
adapting from Waterfall, 40, 45–46, 51–52, 55
staying productive during, 98
standup meetings. *See* daily standup meetings
story points, 31
support tiers (SE)
defined, 102
laying out, 105–106
sustained engineering
about, 101–102
challenges and goals, 102–103
checklist of actions to take, 115–116
collaborating for efficiency, 106–108
consistent vocabulary, 102
determining ownership, 104–105
implementing Kanban workflow, 108–111
Kanban tools, 111–112
key stakeholders, 103–104
laying out support tiers, 105–106
roles and responsibilities, 103–104
troubleshooting problems, 112–115

T

TAR (task add rate), 32–35
task add rate (TAR), 32–35
task completion rate (TCR), 30–35
TCR (task completion rate), 30–35
TDD (test-driven development)
about, 121
evolving from Scrum, 69
sustained engineering, 110
Team Foundation Server (TFS). *See* Visual Studio Team Foundation Server
teams and team members
adapting from Waterfall, 54
capturing high-level routine, 7–8
communicating status to, 91–93
daily standup meetings, 14–17
evolving from Scrum, 58–61
ordering work, 28–29
posting high-level routine on signboards, 8–10
right-sizing teams, 33–37
sustained engineering, 102–105, 112–115
troubleshooting problems, 22–23, 112–115

technical debt
defined, 28
ordering, 28–29
Test-Driven Development by Example (Beck), 121
test-driven development (TDD)
about, 121
evolving from Scrum, 69
sustained engineering, 110
TFS (Team Foundation Server). *See* Visual Studio Team Foundation Server
theory of constraints (TOC)
about, 124–126
drum-buffer-rope, 126–128
This Is Lean (Modig and Åhlström), 135
TOC (theory of constraints)
about, 124–126
drum-buffer-rope, 126–128
tracking
expected completion date, 31–33
work items, 77, 79, 82, 88, 91–93, 111
transportation (Lean waste category), 131
triage team (SE), 106–108
troubleshooting problems
blocked because all items in intermediate step are done, 17–18
blocked because prior step has no items done, 18
bugs impacting team, 20
can't find time to improve tools and automation, 22
constantly getting blocked, 19
important event approaching, 21
item blocked awaiting external input, 20
item needs design work, 21
item needs to be assigned to busy team member, 22
new person joins team, 23
new work, plan changes, updated requirements, 21–22
some team members can't attend standup, 23
some team members like doing more than one item at a time, 22
step taking longer than usual for an item, 18–19
sustained engineering, 112–115
team focusing too much on process details, 23
team has long design discussions during standup, 23

U

unresolved bugs measurement
adapting from Waterfall, 48–51
evolving from Scrum, 62–65
unstable dependencies, 93, 95–97
updated requirements, troubleshooting problems, 21–22
upgrading tools
ordering work, 28–29
troubleshooting problems, 22

V

validate work step
in daily standup meetings, 15
done rule, 13
guidelines, 8–9
troubleshooting problems, 18, 21
value stream, 133–135
Visual Studio Team Foundation Server
continuous deployment and, 82
continuous publishing and, 79
continuous push and, 77
ordering work based on dependences, 88
sustained engineering and, 111
tracking project status, 91, 93
visualization
Kanban and, 123–124
large organizations, 86–88

W

waiting (Lean waste category), 131
Waletzy, James, 101–116
Waterfall, adapting from
celebrating performance improvements, 48–51
checklist of actions to take, 56
completing features before starting new ones, 43–44
dealing with specs and bugs, 44–46
engaging with customers, 46–48
introducing Kanban, 39–42
limiting chaos in, 10
rude Q & A session, 51–56
working in feature teams, 42–43
Xbox examples, 41–42, 47–48

Wideband Delphi method, 30
Williams, Laurie, 122
WIP (work in progress)
adapting from Waterfall, 43–45, 47, 52–53, 55
daily standup meetings, 14–15
evolving from Scrum, 60–62, 66–68
limiting chaos, 10–11
Little's Law on, 31
populating backlog, 25–27
single-piece flow, 124
troubleshooting problems, 17–20
Xbox example, 11
Womack, James P., 135
work backlog. *See* backlog
work in progress. *See* WIP (work in progress)
work-item tracking systems
continuous deployment and, 82
continuous publishing and, 79
continuous push and, 77
ordering work based on dependences, 88
sustained engineering and, 111
tracking project status, 92–93
workflow
bugs and other work, 109–111
critical chain, 129–130
escalations in, 109
implementing, 108

This page intentionally left blank

About the author

Eric Brechner is the development manager for the Xbox Engineering Services team. He is widely known within the engineering community as his alter ego, I. M. Wright. Prior to his current assignment, Eric managed development for the Xbox.com websites, was director of engineering learning and development for Microsoft Corporation, and managed development for a shared feature team in Microsoft Office. Before joining Microsoft in 1995, he was a senior principal scientist at Boeing and a developer for Silicon Graphics, Graftek, and JPL. Eric has published a book on software best practices and holds eight patents, a BS and MS in mathematics, and a PhD in applied mathematics. He is an affiliate professor evenings at the University of Washington's Bothell campus.