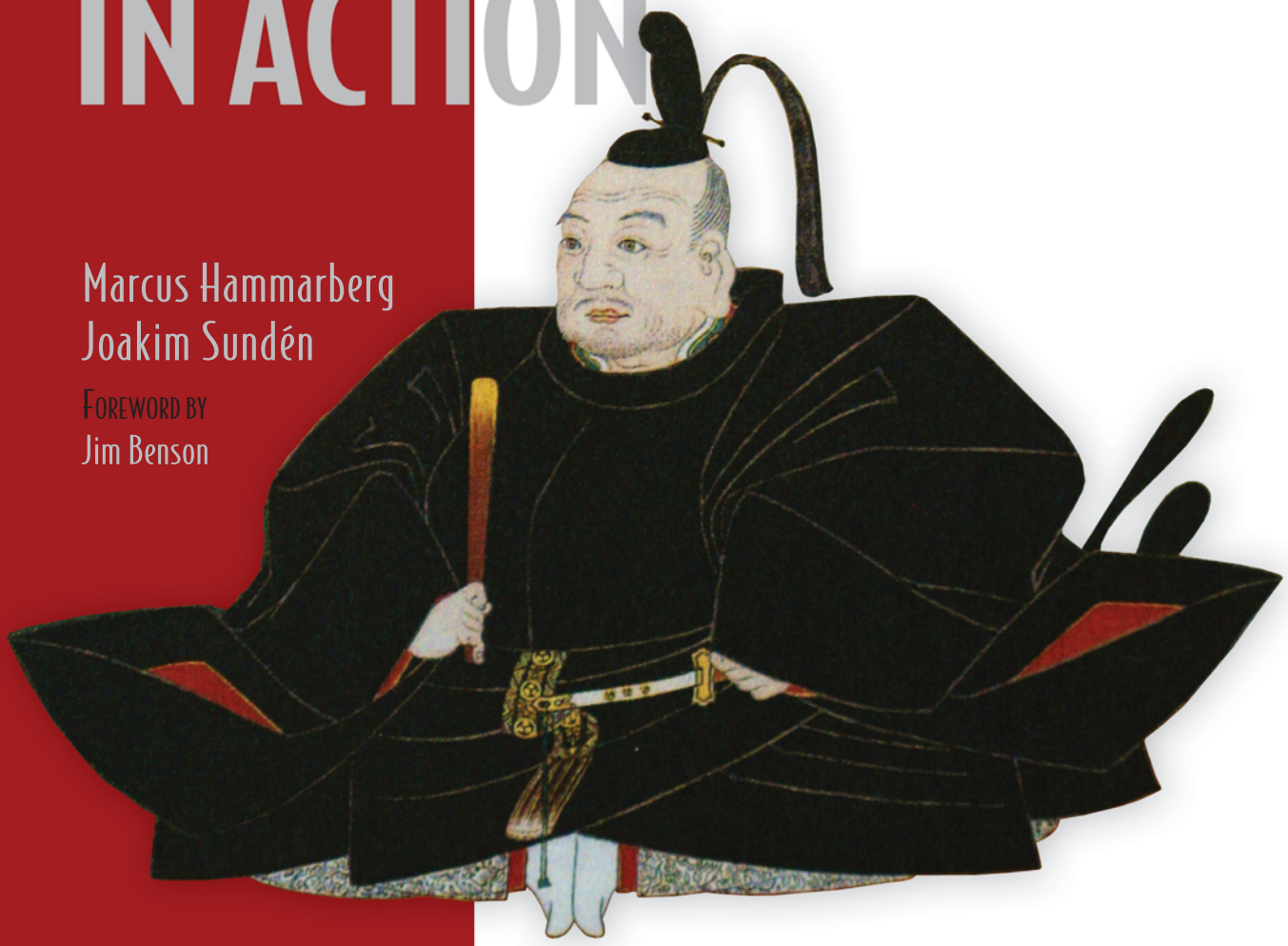
MANNING

Kanban IN ACTION

Marcus Hammarberg
Joakim Sundén

Foreword by
Jim Benson



Kanban in Action

MARCUS HAMMARBERG
JOAKIM SUNDÉN

MANNING
SHELTER ISLAND

For online information and ordering of this and other Manning books, please visit www.manning.com. The publisher offers discounts on this book when ordered in quantity. For more information, please contact

Special Sales Department
Manning Publications Co.
20 Baldwin Road
PO Box 261
Shelter Island, NY 11964
Email: orders@manning.com

Manning Publications Co.
20 Baldwin Road
PO Box 261
Shelter Island, NY 11964

Development editors: Beth Lexleigh, Cynthia Kane
Copyeditor: Melinda Rankin
Proofreader: Tiffany Taylor
Typesetter: Marija Tudor
Cover designer: Marija Tudor

ISBN: 9781617291050
Printed in the United States of America
1 2 3 4 5 6 7 8 9 10 – MAL – 19 18 17 16 15 14

brief contents

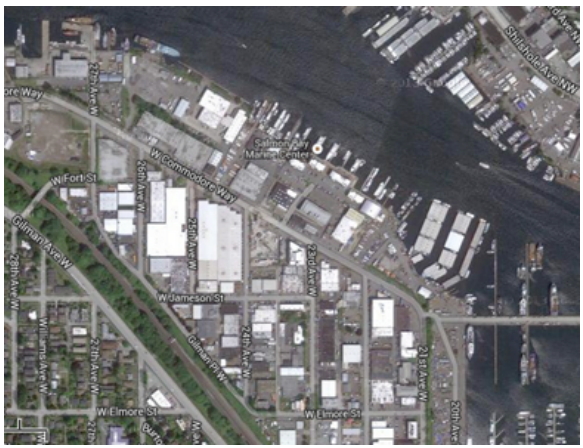
contents

foreword

A great deal of your brain's capacity is devoted to absorbing, processing, acting on, and storing visual information. What we see inspires us to act now and instills patterns for future action. If we have nothing to look at, we have little to act on.

See and understand

Visual systems like kanban draw their power from our preference for visual information. Take a look, for example, at the following simple map. You see the water, the buildings, the roads, and a host of other information. You recognize this immediately. Within the blink of an eye, you understand context, form, and substance.



Here is a list of everything I cared to write down from that map. This is a partial list. And it's in a font size necessary not to fill pages with text:

- Salmon Bay Marine Center
- Lake Washington Ship Canal
- W. Commodore Way
- 20th Ave W
- Gilman Place W
- W Elmore Street
- 21st Ave W
- Gilman Ave W
- Shilshole Ave NW
- W Fort Street
- 26th Ave W
- 24th Ave W

You can quickly see that long lists of things provide less context and take more time to process than a map.

Our goal with visual systems like kanban is to build a map of our work. We want the form and substance of our work. We want to understand the system, immediately and intuitively. We want our kanban board to be explicit about roles, responsibilities, work in progress, rate of completion, the structure of our processes, impediments, and more.

That's a lot of information.

What we've found since launching kanban as a software design tool nearly a decade ago is this:

Seeing the work and the process creates understanding.

Once we see our work, we build a shared understanding of it. Then we can do away with messy process conventions that have plagued software development for years. The kanban board can become a simple single point that lets anyone come and understand the current state of the project.

This means software teams can finally speak the same language as the business! The division between IT and the rest of the company can dissolve. A translator has arrived.

Seeing is half the battle

In this book, Marcus and Joakim list three elements of a project using kanban:

- Visualize
- Limit work in process
- Manage flow

I like this list.

For *Personal Kanban*, we use the first two (visualize your work and limit work in process) and see the third as following naturally. But I like the list of three because it drives this point home:

Work does not fit—it flows.

Smashing work into arbitrary amounts of time has profound negative impacts on rate of completion, escaped defects, and morale. The stress of unnecessary deadlines or overenthusiastic feature sets deprecates both people and product. The focus becomes making work fit into the deadline period, rather than completion with quality.

Completion of work with quality is possible only if work is flowing at a truly sustainable pace. Finding and maintaining that pace is possible only if active work in process (WIP) is less than the capacity of those doing the work. Cramming things in before deadlines will almost always result in breaking your WIP limit.

Too much WIP destroys flow

With a reasonable WIP limit, we encourage the flow of work. Tasks are completed in a measured fashion with an eye on quality. Overhead from managing too much WIP disappears. And, not surprisingly, productivity skyrockets.

This is the short form of what Marcus and Joakim have given you in this book. They provide fantastic and patient detail. If this is your entrée into kanban, welcome. You couldn't have asked for better guides.

JIM BENSON
AUTHOR OF THE 2013 SHINGO AWARD-WINNING
PERSONAL KANBAN

preface

Marcus's journey

I was introduced to agile via Scrum and started to use it, guerilla-style, at a large insurance company in Sweden. Before long, it spread; and within a few years the company had more than 50 Scrum teams. But it still didn't feel right, because the work processes for many teams weren't a good fit with the start-stop nature of Scrum. Also, most teams didn't span the entire process; the teams mostly consisted of developers who were handed requirements and who then delivered to a separate testing phase. I felt the itch to try to incorporate more of the complete process that the work went through.

This itch led me to start investigating other practices in the agile community. Before long, and through some helpful pointers from Joakim, I found and started to read up on kanban. In 2010 and 2011, I attended trainings on kanban and kanban coaching given by David J. Anderson. These further confirmed my feeling that kanban and Lean were what I had been looking for.

Joakim's journey

In 2008, I was consulting as a Scrum Master in a three-team software development project in a large Swedish company's IT department. To deepen my understanding of agile software development, I was reading up on Lean software development—which led me to the amazing story of Toyota and a lot of literature about Lean thinking and the Toyota Way. The studying reached a climax of sorts when I went on a study tour to Toyota HQ in Japan together with Mary and Tom Poppendieck, authors of Lean software development books, in the spring of 2009.

In late 2008, my client came to the conclusion that most, if not all, clients paying for software development eventually draw—that things are moving too slowly. They wanted more development done more quickly, but without cutting scope or quality. Inspired by the Lean thinking around one-piece contiguous flow, I suggested that we should stop planning batches of work in Scrum sprint-planning meetings every two or three weeks (a cadence that felt more and more arbitrary to us) and instead try to focus on one or a few work items and collaboratively get them done as quickly as possible, in a continuous flow of value. The dozen or so team members agreed to not have more than two work items in development and two in testing at any time, and that only when something was finished would we pull new work items from the backlog to plan them just-in-time.

I soon learned about something called kanban that seemed similar to what we were doing, first through Corey Ladas's blog and then through the work of David J. Anderson. In 2009, I connected with the community through the first Lean Kanban conference in the UK. I was immediately attracted by the pragmatic approach of looking at what had actually worked for different teams and companies in their respective contexts, at a time when I felt that a lot of the agile community focus was on faith-based approaches like "How is Scrum telling us how to solve this?"

The next year, I participated in David J. Anderson's first kanban coaching workshop ever (now called Advanced Master Class) in London, together with, among others, experienced practitioners like Rachel Davies, David P. Joyce, and Martine Devos. I cofounded Stockholm Lean Coffee in 2010, where kanban enthusiasts have kept meeting every week since. In 2011, I was invited to attend the first Kanban Leadership Retreat hosted by David J. Anderson, during which I became one of the first "David J. Anderson approved" kanban trainers.

The common journey

Together with our colleague at Avega Group at the time, Christophe Achouiantz, we started developing a practical introduction to kanban in 2010. It was an immediate success and the starting point for a long series of conference talks in both Europe and the US, including in-client trainings, tutorials, and workshops, sometimes conducted individually, sometimes by the two of us together. The practical approach of our work resonated well with many people who attended our talks and tutorials, and we received a lot of positive feedback.

It was after a conference tutorial at JFokus (a great conference organized by Mattias Karlsson, another Avega Group colleague) that Marcus got a call from Manning Publications, asking him if he was interested in writing a book. He immediately felt that he should do it together with Joakim. We decided to write the book in the same manner as the presentation we had created, using a practical approach and a lighthearted style.

about this book

Do you want to better understand how your work works and what is happening on your team or in your workplace? Would you benefit from being able to focus on a few small things instead of constantly having to switch between multiple projects? Do your users and stakeholders want new features delivered now rather than some other day? Do you think that you and your coworkers need to keep improving and learning?

Then *kanban* is for you.

Do you want to get started with kanban as soon as possible, without spending too much time on abstract theory and history and splitting hairs about different methods? Do you want to know how people in the kanban community have used kanban in practice to face different challenges?

Then *this book* is for you.

This book is a down-to-earth, no-frills, get-to-know-the-ropes introduction to kanban. It's based on lots of practice, many observations, and some hearsay (!) from two guys who have worked with and coached dozens of kanban teams. We've also talked and taught at conferences and actively participated in user groups and the kanban community over the last few years.

In this book, you'll read about simple but powerful techniques to *visualize work*: how to design a kanban board, how to track work and its progress, how to visualize queues and buffers, and even such nitty-gritty details as how colors and other enhancements can help you to organize and track your work items.

You'll also pick up a lot of practical advice about how to *limit your work in process* throughout the workflow, such as how to set the limit in different ways depending on context, and how to understand when and how to change it.

With these two tools in hand—kanban and this book—you're ready to get down to business and *help your work flow* through the system as you learn and improve your process further and further. You'll learn about things like classes of service, how planning and estimation are done in kanbanland, about queues and buffers and how to handle them, and—well, you'll learn a lot of things that you'll need to help your team become a little better every day.

But wait, there's more. You'll learn about metrics and how to use them to improve, and we'll present several games and exercises you can use to understand the principles of kanban and get new people to join you on the kanban bus. Hey, we even throw in a small section on kanban pitfalls and common criticisms, just for good measure.

This is a practical book, and we won't spend a lot of time on the underlying theory or the history behind kanban. There are already great books on these topics (hint: pick up some books about Lean, agile, and Toyota), and they do a much better job at that than we could ever dream of doing. But we won't leave you high and dry; some theory will be needed to make good use of the practical advice we're giving, and we'll supply it to you.

But this book is not only for beginners. Judging from all the questions we receive about kanban, and from all the light bulbs that get turned on during our practically oriented talks and training courses for people who have been working with kanban for some time, as well as for novices, you'll get a lot out of this book even if you're far from new to kanban.

Let's get started and see some *kanban in action*!

The structure of this book

This book is divided into four parts, each with a different purpose, aimed at being your companion as you learn kanban:

- *Part 1, "Learning kanban"*—This is an introduction to kanban in the form of a short story. The idea is that you can quickly skim through this part to get a feeling for what kanban is and learn enough about it to get you up and running, just like the fictional team you'll meet in chapter 1. After this introduction, you'll have all the tools and knowledge you need to start using kanban in real life—you'll be able to start learning by doing kanban. If stories aren't your thing, or if you don't like our storytelling style, you can skip this chapter and jump straight into the next part.
- *Part 2, "Understanding kanban"*—This part gives you deeper knowledge about the *why* (the principles and ideas behind kanban) and the *how* (lots of practical tips on applying the principles in your context). We'll take a closer look at the core principles of kanban. There will be many commonly used solutions and variations on these, which people in the community have applied in different contexts. Our descriptions will be practical and will give you more tools and tips to continue to build your knowledge. The team from chapter 1 will pop in from time to time to ask questions.

- *Part 3, "Advanced kanban"*—OK, you're up and running with your board, you're familiar with how WIP limits work, and you're focused on helping the work to flow. Now what? In chapters 8–12, you'll learn how to use kanban principles to manage risk, facilitate self-organization, plan, and improve. We've also included a chapter on common pitfalls and how to avoid them. Don't let the "advanced" scare you: it's not that complicated, it's just that these practices aren't what you start with typically when you're new to kanban.
- *Part 4, "Teaching kanban"*—If you start using kanban in your organization, you'll soon find yourself teaching kanban to others. We have found it beneficial to do this through games and simulations, because some of the principles seem counterintuitive at first.

We make no claim that you'll come out a kanban master at the end of this book, but it will make a good companion on your learning journey. Matched with the practical experience you'll gain from trying stuff out, this will be a great learning combination.

How to read this book

You can choose several ways to read this book:

- *If you want to get started as fast as possible,* spend an hour reading part 1 ("Learning kanban"), and implement some of the things you learn right away.
- *When you need inspiration or get stuck,* browse through part 2 ("Understanding kanban") and steal ideas or be inspired by how others have approached similar challenges.
- *If you want to know why things are how they are in kanban-land,* read part 2 and learn where kanban comes from and the principles and ideas on which it's based. You'll get a hefty dose of practical tips along the way.
- *If you're already using kanban and are curious about the next step,* take a closer look at the topics in part 3 ("Advanced kanban"). You'll be sure to pick up something new that applies to your situation.
- *When people ask you to teach them kanban,* find fun and educational games in part 4 ("Teaching kanban") to play with them, and tell them about your findings and experiences. And then get them a copy of this book!

You can also read the entire book from cover to cover. This will give you a gradually deeper and wider understanding of kanban. We believe that the best learning experience will come from combining the topics in this book with practical experience.

Author Online

Purchase of *Kanban in Action* includes free access to a private web forum run by Manning Publications where you can make comments about the book, ask technical questions, and receive help from the authors and from other users. To access the forum and subscribe to it, go to www.manning.com/KanbaninAction. This page provides

information on how to get on the forum once you're registered, what kind of help is available, and the rules of conduct on the forum.

Manning's commitment to our readers is to provide a venue where a meaningful dialog between individual readers and between readers and the authors can take place. It's not a commitment to any specific amount of participation on the part of the authors, whose contribution to the forum remains voluntary (and unpaid). We suggest you try asking the authors some challenging questions lest their interest stray!

The Author Online forum and the archives of previous discussions will be accessible from the publisher's website as long as the book is in print.

about the authors

Before we set out on this journey together, it might be interesting for you to get to know us a bit. Here we are—plain and simple:

JOAKIM is a thinker, the brains in our dynamic duo. He often lets a person talk for quite a while before he makes up his mind what to say, and then he responds with something profound meant to make them think. This annoys some people, because they usually just want to know what "to do." He has solid theoretical knowledge in all things Lean, agile, and about the Toyota Production System. And he has a lot of practical experience to go along with it, too.

In his spare time, Joakim is a foodie and a movie buff, and quotes from obscure Danish dogma movies sneak into his conversations from time to time (much to the confusion of those around him).

Joakim has four kids (ages zero to nine) and a wife (Anna) and still manages to be engaged in the progress of the company he works for (Spotify) and the Lean and agile communities in Sweden and around the world. He's a regular speaker at international conferences.

MARCUS is a doer and thus the muscle of the pair, to continue with the "dynamic duo" metaphor. He prefers to try something out and fail rather than think about doing it right the first time. This leads to him having do stuff over and over again—much to his irritation and the amusement of others.

Marcus has approached the Lean and kanban communities from a developer's perspective and has a strong interest in the

practices that make these ideas work in the wild: test-driven development, pair programming, specification by example, and impact mapping, among others.

When he has time, he can be found blogging or at the Salvation Army or reading up on the latest brass-band news. Trying to incorporate much of that into work-related situations is both hard and pretty much useless, as you can probably imagine.

Marcus is married to Elin, and they have three boys (5, 3, and 3 years old[1]). By the time you read this, they will all have moved to Indonesia, where Marcus will work for the Salvation Army. He will lead the work at a foundation, for the Salvation Army's 6 hospitals and 13 clinics in Indonesia. This will, of course, be done in an agile, Lean fashion, drawing inspiration from and using the techniques found in this book. Marcus will also teach brass instruments to the youngsters at the Salvation Army orphanages.

[1] Yes, the last two are twins.

about the cover illustration

The figure on the cover of *Kanban in Action* is Tokugawa Ieyasu (1543 – 1616), the founder and first shogun of the Tokugawa Shogunate of Japan, which ruled from the Battle of Sekighara in 1600 until the Meji Restoration in 1868. A shogun was the military leader in feudal Japan, and because of the power concentrated in his hands, he was the de facto leader of Japan, in place of the nominal head of state, the mikado or emperor. Ieyasu seized power in 1600, received appointment as shogun in 1603, abdicated from office in 1605, but remained in power until his death in 1616. He claimed to have taken part in over 90 battles during his lifetime, as either a warrior or a general. He had a number of qualities that enabled him to stay in power and wield authority—he was both careful and bold—at the right times and in the right places. Calculating and subtle, he switched alliances when he thought he would benefit from the change.

We would like to share one of Ieyasu's recorded quotes with our readers, a quote that is applicable to both our personal and professional lives: "Life is like unto a long journey with a heavy burden. Let thy step be slow and steady, that thou stumble not. … Find fault with thyself rather than with others."

acknowledgments

If you've read an acknowledgements section before, you know that it always starts with thanking the families of the writers. We now know why. They are the people from whom we have taken time: writing while they fall asleep, writing instead of spending time with them, giving them cryptic answers when we were somewhere on page 267 instead of at the playground where we should have been. And *still* they supported us throughout this project. Without them and without their support, this book would not have been possible.

We owe the community around us a big thank you for this opportunity—all the people we have learned from, and continue to learn from, every day and who in many cases know this stuff better than we do. We're standing on the shoulders of giants. Thanks for your shoulders and your encouragement during this process.

There are other people we want to mention who have been particularly helpful, inspiring, and supportive: Christophe Achouiantz, Torbjörn Gyllebring, David J. Anderson, Jim Benson, Corey Ladas, David P. Joyce, Benjamin Mitchell, Karl Scotland, Mattias Skarin, Don Reinertsen, Alan Shalloway, Mary and Tom Poppendieck, Håkan Forss, Måns Sandström, Eric Willeke, Jabe Bloom, Mike Burrows, Dennis Stevens, and all the folks at the Kanban Leadership Retreat. We've learned a lot and had a great time with the Stockholm Lean Coffee bunch, including Håkan Forss and all the other wonderful people there.

An array of people also helped us with reviews and feedback, for which we are very grateful. A special thank you to Rasmus Rasmussen and Viktor Cessan for your insights, and to the following reviewers: Adam Read, Barry Warren Polley, Burk Hufnagel, Chris

Gaschler, Craig Smith, Daniel Bretoi, Dror Helper, Ernesto Cardenas Cangahuala, Jorge Bo, Karl Metivier, Marius Butuc, Richard Bogle, and Sune Lomholt.

Special thanks to Jim Benson for providing the foreword to our book, to Danny Vinson for his careful technical proofread of the manuscript shortly before it went to production, and to Robert Vallmark for producing the great-looking[2] avatars—you really helped us improve the book's visuals!

We have been fortunate to work together with the great crew at Manning, and we are convinced that Manning set aside their best people just for us.

Thank you to Bert Bates for helping us push the envelope on how a Manning book could look and feel. We're fortunate to have had access to your head at the beginning of this process. And of course, thank you to publisher Marjan Bace for letting us write the book this way.

A big thank you to Beth Lexleigh and Cynthia Kane, our development editors, for your effortless reviewing and pushing when things were slow. You took our ramblings and turned them into a real book.

Thanks to all the other people at Manning who helped us in ways big and small, in no particular order: Michael Stephens, Maureen Spencer, Tiffany Taylor, Kevin Sullivan, Mary Piergies, Janet Vail, and Candace Gillhoolley.

MARCUS

I first want and need to thank God—the foundation of everything I am and do.

My personal thank you goes to Elin and the boys (Albert, Arvid, and Gustav), who have supported me during this process. I even got some design help from Albert from time to time.

To my father and mother who raised me to be what I am today (for better or for worse): "Tack mamma och pappa, för allt ni gjort för mig."

To all the people in my close community whom I've turned to with questions and worries from time to time—a mega thank you. I got nothing but cheering and support from you guys: Torbjörn Gyllebring, Håkan Forss, Måns Sandström, Anders Löwenborg, Hugo Häggmark, Tomas Näslund, Per Jansson, Kalle Ljungholm—love you guys.

To Avega Group and Aptitud (my employers during the time of writing): thank you for letting me take on this project. Avega even paid me for it! It blew me away, when you offered that! You're great, both of you!

And finally, to Joakim—this book would have been rubbish without you. It might have been finished earlier, but no one would have read it. I've learned more than a lot from you and continue to do so. Thanks, man!

[2] Great looking and funny caricatures, although not very flattering to us. Joakim's avatar received the comment "It looks like an Italian version of you after you've had too much pizza," and Jim Benson asked why Marcus looked like Jeff Goldblum.

JOAKIM

My participation in this book would not have been possible without the support from my family. To the family that gave me life and amazing opportunities (in chronological order): my grandparents Albin, Ingeborg, and Molly; my parents Ove and Elisabet and their siblings and their families; my sister Anna and my brother Henrik—thank you for making me who I am. I'm extremely grateful to the love of my life, Anna, and our children Alva, Saga, Albin, and Iko—your support has been phenomenal, as always.

Thank you Marcus for involving me in this book; for your endless patience with my slow and sparse contributions; for constantly soldiering on and writing; for coping with my not always very polite criticisms, recommendations, and ideas for big rewrites to be mainly carried out by you; for the huge effort you've put into the drudgery of formatting, pixel-pushing, and so on; for pushing me yet never rushing me or making me feel bad for not contributing enough (I did that myself); for your cheerful and supporting personality; in short, for being Marcus!

Part 1

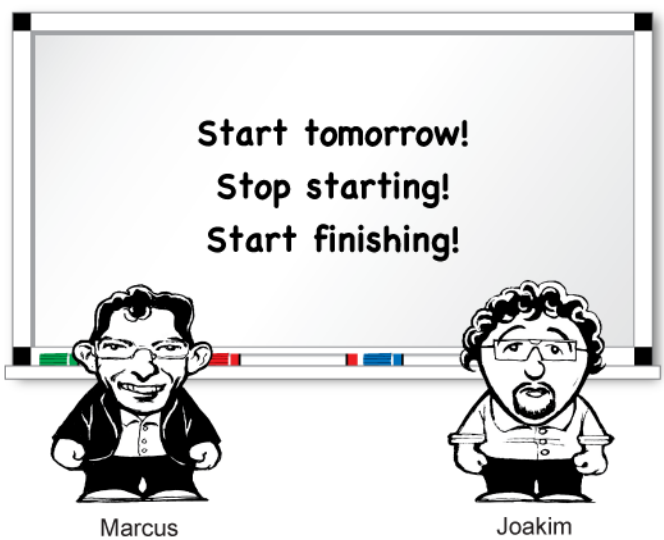
Learning kanban

Part 1 is a practical introduction to kanban. The goal of this part is to enable you to get up and running using kanban while also giving you a basic understanding of the principles behind it and peeking into some advanced topics to whet your appetite for more.

We start off with a short story that follows a typical software development team as they are introduced to, and get started using, kanban. If you don't like the story-telling approach, you can skip straight to the next chapter; we'll cover most of the things from chapter 1 in more detail in subsequent chapters.

1 Team Kanbaneros gets started

Marcus and Joakim are at a conference presenting a practical introduction to kanban. They're finishing the presentation; let's join them in action.



"To sum up: kanban is an approach to software development based on the principles of Lean. It has quickly been picked up by many organizations around the world. You can pick it up too! Starting tomorrow, you should stop starting and start

finishing. And with that," Joakim concluded, "our quick, practical introduction to kanban ends. But remember what we said earlier—you could get started tomorrow. Getting up and running with this isn't hard—the effects can have a dramatic impact on your productivity."

"Thank you all for coming! We'll be hanging around here for a couple of minutes if you have any questions," Marcus added, trying to close the discussion in order to end the presentation on time—for once.


Daphne

As Joakim started to clean the whiteboard and remove all the stickies from the wall, Marcus answered a couple of quick questions, pointing some people to the slides available for download as he headed toward the exit. He didn't get far, though. Halfway to the exit a woman abruptly stopped him.

"What's this? Why are you leaving? Don't tell me we missed it all?!" The woman looked disappointed and almost as though she'd been cheated.

"Missed what? The presentation? Yes, it just finished, but you can catch the video online," Marcus replied.

"Oh no!" she cried out. "We only came here for this presentation! It looks like *someone* missed the starting time of the presentation." She nodded toward a man leading a group into the room.

"Well, we'll probably do something in the autumn as well if you want to see the live presentation," Marcus said, trying to smooth things over.

"That won't cut it—we want to get started right away! Our whole team is here; even the business guys were joining us for this tutorial, on my recommendation." She looked genuinely sad. "Hi, I'm Daphne, by the way."

"OK; why don't you book Marcus or me for a day of consultation, then?" Joakim suggested as he joined them, introducing himself to Daphne.


Cesar

"Well, we could, but there have been quite a lot of complaints from both the team and people working with it already. We wanted to do something about this and hoped that we would pick up something to help today." The words came from an imposing man who had caught up with Daphne.

"What kind of problems are you talking about?" Joakim asked the man.

"The team feels swamped with work, and people who are waiting for them to deliver stuff thinks it takes forever," he said plainly. Daphne cut in: "Despite us having a lot of work to do, we have long discussions about which project is more important to start first. Yet we still fail to pick the right one."

"And … well, there's more, but we don't want to take up your time. You were leaving, right?" the man said, leaving the question hanging.

"Unless you have a better suggestion?" Marcus replied quickly. This sounded like the beginning of a fun challenge.

"This is what we're going to do. I'll buy some consultation from the two of you right here and now," the imposing boss-man said. "Your slide there on the screen says 'start tomorrow.' I'm giving you a chance to put your money where your mouth is. How much time can you spare?" He paused and locked his eyes on Marcus and Joakim.

"We have two hours until we need to get going," Joakim said, looking at his watch.

"Well in that case: get us up and running with kanban in two hours," the boss-man said, extending his hand.

"We won't be able to help you solve all your problems, only put you on the right path," Joakim said, looking at Marcus. They nodded to each other. "Right—we have two hours to spare. Challenge accepted!"

Welcome to kanban in action!

You've embarked on a learning journey about kanban, and this introductory chapter will teach you the basics of kanban by means of a story. No previous knowledge of kanban is needed. You'll follow coaches Marcus and Joakim (yes, that's us) as they teach kanban to a software development team and help the team apply what they learn to their way of working.

This made-up story is meant to be an introduction that easily and gently takes you through the basics of kanban. In this chapter, you'll learn how to use visualization techniques such as the kanban board and its work items to better understand how your work *works*. You'll learn how to limit the amount of work in process, and you'll come to understand how doing so makes your work flow faster and helps you identify improvement opportunities. You'll also learn a bit about using metrics for improvement.

In the reviews we've gotten on this chapter, there have been two camps. Many people love this storytelling approach; others seem to … not like it as much. If you don't want to start with a story, you can skip to the subsequent chapters directly. Everything we mention in this chapter will be treated in much more depth in later chapters. But we hope you'll read this chapter, too, and get a lot out of it. After reading this chapter, you should be able to get up and running with kanban yourself, by applying the principles we're teaching the team in the story.

In the rest of the book, we'll go into the details: the principles behind kanban and a lot of variants and practices that kanban teams around the world have evolved. If you feel that we've left you with questions in the first part, you'll certainly find the answers in the latter parts of the book.

But first things first; let's get back to the story!

1.1 Introductions

The entire team was gathered in the next room. They introduced themselves to Marcus and Joakim by describing each other, a technique they seemed to be pretty experienced with, judging from the bold statements made.


Adam

"Adam is a tester. He's been around for quite some time and is almost always skeptical: of new stuff, of the others' capabilities, but mostly of the quality of the work the others do. He tries to deliver his criticism in a nice way; sometimes he succeeds."

"Adam likes to work his way. He doesn't like change—change means regression testing."


Beth

"Beth is a new employee. She's responsible for requirements analysis on the team. That includes everything from asking the business what it wants and writing it down to making sure the developers understand what they should do."

"Beth is always looking for new ways to work."


Cesar

"Cesar *is* the business. He practically built the first version of the application the team is working on, an internet bank, all by himself way back when."

"Now he has left the IT part of things and is in charge of the business part of the operation. He still likes to think of himself as a developer and is often found 'on the floor.'"


Daphne

"Daphne is a kick-ass developer. She has been known to sling more code in an hour than most developers do in a week."

"But she likes to work alone; other people slow her down. Sitting down with Cesar and deciding how stuff should work—that's the best way to work, if you ask Daphne. Things fly out then. All these other ceremonies and hierarchies are in her way."


Eric

"Eric is a developer by day … because he has to be. But at night he's a guitar hero, playing in local pubs and other venues. Soon the big break will come. Soon."

"Writing code is all right, but it's often hard to see why we do it."

"Eric likes to get things done so that he can continue to answer questions on http://guitar.stackexchange.com."


Frank

"Frank is the manager of the IT side of the operation. He has 36 people under him, and he tries to meet with everyone at least once every other month, but there are lots of other meetings to attend."

"He cares about the product, but he has a hard time keeping up with all the new features. Frank tries to develop his people whenever he has extra time."

"Great," Marcus said. "I've found a room over here with a whiteboard in it. Do you have your stickies, Jocke?"

"Yes, of course," Joakim replied, with his please-no-stupid-questions face. He's an agile coach; naturally he has stickies with him at all times.

"Great—let's go, then," Marcus said, leading the way.

"Who can tell us what your team does?" Joakim asked when they stood around the board.

"That's probably me," Cesar said, pulling out his laptop.

"No, please! No slides!" Eric cried out in pain. "Tell him—we don't have time for that."

"Yes, you're probably right," Cesar responded, a bit stumped. "Here's the short version."

The team is a small development team consisting of the group gathered here. They work for a big insurance company with responsibility for the newly released mobile bank application. Because their team is pretty small, they can govern themselves for the most part. There's reporting to be done to other parts of the business, but they can make their own decisions about the work they do, with Cesar having final say in every important decision. The team is in charge of creating new features in the mobile bank, as well as supporting and maintaining the software that's in production.

"Why do you need our help?" Marcus asked. While waiting for an answer, he wrote the heading *Challenges* on a flipchart.

"I can probably answer a bit of that." Frank, the project leader, stepped forward. "We've been experiencing difficulties in keeping up the pace of expected deliveries. There has been a lot of complaining from different stakeholders in the organization about not getting their features in time."

"And that gets worse by the minute," Cesar added. "People no longer trust the quality of our work, and they definitely don't trust our estimates and delivery dates."

"We, on the team, for our part," Daphne added, "feel totally swamped and don't know what to do first. When we're trying to please everybody, it leads to sudden changes in priorities, and everything is 'PRIO 1.'"

The team then told Marcus and Joakim that it was common for stakeholders to hand tasks directly to the team members. These requests often came from senior people in the organization, making it difficult for the developers to say no. Furthermore, those items were sometimes tasks someone else was already working on. Marcus patiently wrote down the challenges as bullet points on the flipchart.

* We often deliver late
* Estimates are often inaccurate
* Team is swamped with work
* Priorities are unclear
* Work is coming to the team from everywhere
* Unclear who's working on what

"Thank you, but that's probably enough," Daphne interrupted. "The clock is ticking, you know; you only had two hours to spare, right? Let's get practical."

"Yeah, let's get back to the task at hand." Frank grew impatient now. "Where do we start?"

"We understand that you're eager to get started," Joakim answered. "But you should also understand that kanban is a bit different from other methods. Take Scrum or Rational Unified Process (RUP), for example; they prescribe what roles you should have, what meetings you should run, even how you should run them, and so on. Kanban, on the other hand, starts where you are, helps you understand your current situation, and helps you identify the next step to improve it."

"Yes, it's important for us, and for you, to understand where you are right now in order to help you," Marcus cut in, afraid that Joakim would come across as too theoretical, "but we can definitely get started now, and you'll understand more about this as we go. We'll keep this list of challenges here as a sort of agenda for our short time together."

"What is your team called, by the way?" Joakim asked.

"From now on it should be: the Kanbaneros!" Frank said, in his best Mexican accent. The rest of the group laughed, and Marcus wrote it down on the flipchart.

"Let's start with some show-and-tell to make our work a little more visual, shall we?" Joakim began.

1.2 The board

"How do you know what you're working on as a team and how work enters your workflow? If you're not even sure yourselves, how could the stakeholders know, right?" Joakim went up to the whiteboard and grabbed a marker.

"Where do you keep your backlogs today?" Joakim asked. "I guess you have some kind of list of what you need to work on?"


Beth

"Of course we do," Beth quickly replied. Joakim and Marcus could sense that she felt a bit insulted by the question. "I make sure to enter and categorize all the requirements in JIRA,[1] our project-issue tracking system."

"Yes, and I go in there as often as I get a chance and try to keep the ordering, with respect to priorities, up to date." Frank added that he felt the project-tracking system was in good shape.

"This makes it easy to see who is working on what and the progress for each item," Cesar added.

"Can we access your JIRA system right now?" Marcus asked, pointing to Daphne's laptop.

"Yeah—of course," Daphne answered, and she flipped the laptop open.

"Good. Let's write down what you're working on right now," Joakim suggested. "Write each item on a separate sticky note, and keep it brief; it's enough if all of you understand roughly what it refers to." He opened the pack of sticky notes with his patented one-hand grip in less than a second, much to the amused admiration of the group (or so he imagined). He handed out yellow notes to the team.

"Please write legibly, using the thicker sharpies and not the ballpoint pens." Even though Marcus had given the exact same instructions hundreds of times, he always felt a bit patronizing when doing so, but he had learned the hard way that tiny scribbling makes it much more difficult for people to understand and feel involved in exercises like these. "When you're done, post the items on the whiteboard."

Joakim asked the team to post the work in priority order from top to bottom. There were a couple of minutes spent running back and forth to the screen, but pretty soon six items had been posted in a nice column on the board. When the running had settled down, Joakim faced the team and asked, "Do you agree, team? Is this what you're working on right now?"

"Yes," came the answer, almost instantly and simultaneously from Cesar, Frank, and Beth. Adam and the developers, Daphne and Frank, looked down at the floor and said nothing.

"There's no right or wrong here," Marcus said. "What's important is that we get the true picture of your current situation. Anything else?"


Adam

After a couple of seconds, Adam came clean. "Well, it's not entirely true … we have a lot of stuff that we do that doesn't get entered in JIRA."

"What are those items?" Joakim asked. "Can you give us examples?"

It turned out that those items could vary a lot both in size and in what they were: additional requirements, small support tasks, favors repaid to other departments, and maintenance of

[1] JIRA is an electronic issue- and ticket-tracking application from Atlassian: www.atlassian.com/software/jira/overview/.

systems were some examples. Often, someone senior who wanted something done handed it to a team member in the corridor.

"Those tasks are hard to turn down," Adam said. "We don't know how to prioritize or what we're allowed to say no to. But I guess we could do a better job telling you about them."


Beth

"Well, to me it seems that we're missing JIRA tickets," Beth said. "If people are working on stuff that isn't registered in there, then we can't trust JIRA."

"Is there anything you could do about that?" Joakim continued.

"Well, I guess it will sort itself out if we make sure to add everything to JIRA, right?" Beth said. "Then all the work will be in one place, and it's easy to share with people who don't sit next to us."

"That's right," Joakim said. "Any drawbacks to an electronic tool?"

"Not where I stand, no," Eric was quick to say.

"Hmmm—things often get lost in JIRA," Beth commented. "We have, on more than one occasion, had doubles of work, for example. There's a fair amount of searching to find anything in there."

"How could we make work easier to find and see?" Joakim asked.

"Well," Beth began, "we would have to see it all the time, I presume. Like writing on the wall or something."

"Bingo!" Marcus could not hold back anymore. "Put the work on the wall! For every work item you're working on, create a little note and put it up on the wall. You'd be surprised how big a difference such a small thing will make for your work." Marcus knew from experience that it was a big thing for many organizations that were starting with agile development.

To see this in practice, Marcus suggested that the team create a sticky for each work item they were working on right now and then post them on the board.

"Everybody done?" Joakim asked after a minute or two. "Please post them on the board."

Next to the previous six items from JIRA, another eight were posted.

"My God!" Frank cried out. "Is this true?"

"We told you!" Daphne said, throwing her arms out wide. "We get a lot of stuff on the side."

"I know, but I never realized …" Cesar scratched his head. "You've been through some bad, stressful times, people."

"But that's more than the *actual* work in JIRA," Beth said, as she counted the stickies.

"That's what we've been saying all along," Eric said, changing the tone of his voice for once.

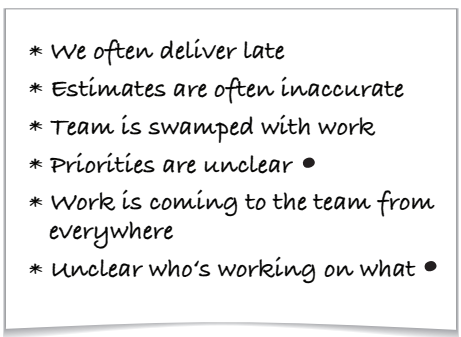
The group grew silent and gazed at the board for a while. By creating small notes that represented each work item, they had shown, in a physical way, what they were doing. Joakim and Marcus had witnessed this eye-opener for teams many times before.

"There you have information being picked out of JIRA, the electronic tool—or the 'information fridge,' as you like to call it, Joakim," Marcus said. He looked at Joakim, pausing to have him explain the metaphor.


Joakim

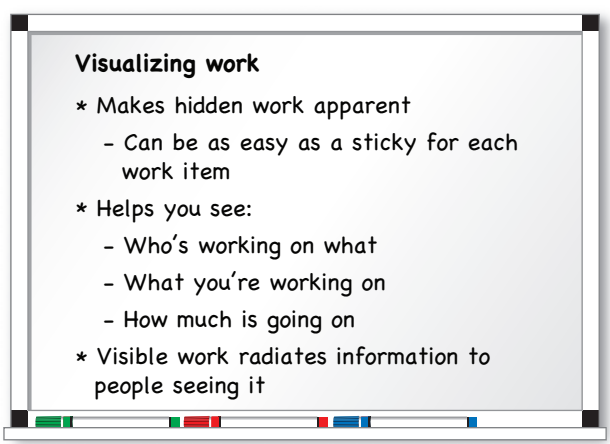
Joakim picked up the cue and explained: "A visualization on a big visible board can be called an *information radiator*; its information is obvious and apparent to people passing by. Electronic tools are, I think, sometimes more like fridges in that you have to open them and poke around to find what you're looking for." He wasn't too sure anybody heard him, as the team stared at the pile of stickies in front of him.

"Looking back at your challenges," Marcus intervened, "we've at least started addressing the confusion of who's working on what and what work the team is actually doing, and we've touched briefly on prioritization."



Beth took in the scene before her with the stickies on the board. After a minute she said, "What do we do with that pile of stickies then?"

"Yeah, stacking them on the wall doesn't help us get more work done and keep the stakeholders satisfied, does it?" Daphne added.



1.3 Mapping the workflow

"Well, all those stickies show you that you have a lot of work in progress," Joakim started. "But there is more you can get out of visualizing your work, like letting everyone see the progress of the different work items and what the priorities are. This will further improve transparency and help you to prioritize work as it comes to you," he said, pointing to the list of challenges. "Let's try to map your workflow to the board and put the stickies in the right places." Joakim removed the cap from his whiteboard marker, ready to go at it.

"Map our what to the what, now?" Beth asked.

"Well, the items you work with *flow* through your system from idea, or customer request, to finished feature in production, providing value to the customer, right?" Joakim asked.

"Yes … I guess so." There was mumbling all around.

"Where does the work start?" Joakim asked.

"Hmmm—hard to say," Frank said. "It depends on where you draw the line. Is it when the idea is first conceived? Or when we decide to do it, or get the approval to do it? But I guess something that's concrete is that we register all the stuff we're doing in JIRA, our ticket-handling system."

"But Beth," Cesar interrupted, "you and I have done a lot of work before that, right?"

"Yes," Beth replied, "but that's a bit unstructured, because it goes back and forth a lot before we decide what to put into JIRA. Sometimes it's a big idea, and sometimes it's a little fix."

They decided to start the workflow when the work was registered in JIRA and to end it when the work was in production.

"I can't see why we shouldn't track it all the way," Daphne said.

"OK, let's draw the workflow as columns that the work progresses through." Joakim started drawing columns on the whiteboard, but then he stopped. He had tried too many times to draw the board for a team, and knew that it would make the board *his* rather than the *team's*. Luckily there was a simple solution. "Better yet; you draw them." Joakim threw the marker into Beth's lap.

Marcus

"To indicate that some work is in a certain stage, we put the sticky in the column with the corresponding name," Marcus explained.

"But what are the names or stages, then?" Beth looked confused, because Joakim hadn't written any names.

"Well, that's where you guys come in," Joakim said with a wry smile. "We can't do all the work for you. The stuff you haven't done yet in JIRA—what do you call that?"

"Well … I don't know. Undone?" Frank said. There was laughter all around.

"How about Inbox?" Cesar suggested.

"I don't know," Eric said. "My thoughts go to the mailbox. And I hate email. Can't we try something else?"

"To do," Daphne suggested. "Or is that too simple?"

Nobody seemed to mind, and Joakim said: "That's a great start! Remember, we want it simple and easy. If you see a need to later, you can always change it."

"In fact," Marcus said, raising a finger, "we suggest that you draw the board with a marker on a whiteboard. Don't make it permanent, using tape and stuff for the columns too soon, because that might make you less inclined to change it. And you'll have to change it as you learn."

As Marcus talked, Beth drew a column on the board to the far left. She wrote *Todo* at the top of the column.

Joakim said: "OK, all the stickies that you haven't started working on yet—move them into that column."

The team went up to the board and started to move items around. There were some discussions and questions about what items actually meant, but pretty soon three stickies were posted in the Todo column.

"OK, guys," Joakim called out to the group as they huddled around the board. "Let's make it easier from the outset, and put the items in prioritized order, from top to bottom. This will be helpful later on, because you'll be able to easily see what is more important in each column."

"How do you go about working with the item from there?" Marcus asked. "What is the first thing you do?"

"We often sit down for a design discussion. How should this be built? What can we use from our existing platform? Should we use other teams' components for the new solution? Things like that," Frank said.

"There's some documentation of our findings, as well," Beth added.

"What would you call that?" Joakim asked.

"How's Design as a name?" Cesar asked.

"Hmmm, not quite right. There could be a lot of different things to do. Maybe Analyze would be a better fit? What do you think?" Frank asked Marcus.

"I haven't got a clue. It's *your* workflow. Ask your team instead. What do you all think?" Marcus faced the group.

"Yeah—Analyze is better," Daphne said, and the others nodded and murmured in agreement.

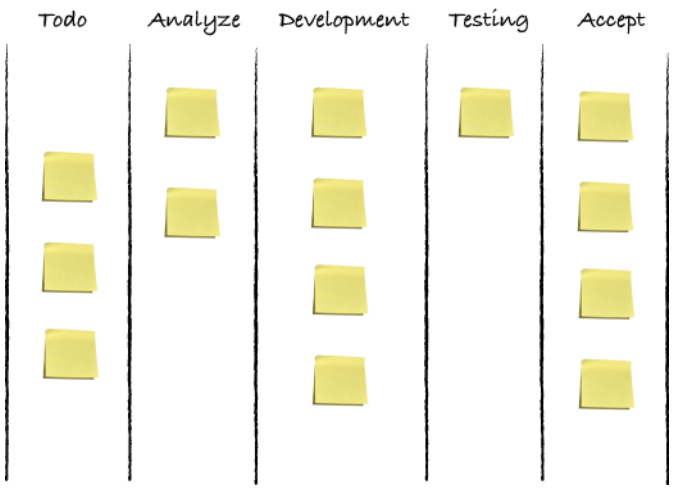
"Anything you're analyzing right now?" Marcus asked.

The team agreed to put two items in the Analyze column.

"Continue down your workflow, following the flow to 'in production.'" Joakim thought that Lean terms, strung together, sometimes became poetry. A quick look around confirmed that he still was alone in that belief. He coughed slightly and said, "OK, what happens after that?"

The team continued by creating columns for Development and Testing, and they added the stickies with the current work to the columns. After that, there was a brief discussion about what "done" meant. They decided to add a step in the workflow for acceptance. They settled on Accept as a column name. When asked, Adam also realized that three of the items in the Testing column were tested already and were waiting for Cesar's approval. They decided to move those into the Accept column as well, together with another item that was already there.

Adam went up to the board and moved the stickies from Testing to Accept.



"Then what?" Joakim said. "You have your work analyzed, developed, tested, and accepted. What more needs to be done?"

"Well, we should roll it out in production, dude," Eric said.

"And by 'we,' you mean …" Daphne prompted.

"Ops." Eric shrugged, sighing.

"What's that?" Marcus said.

"Well, that's operations," Frank, ever the diplomat, explained. "The operations part of our company has been outsourced to another company. That's increased the time it takes to get a deployment into production."


Eric

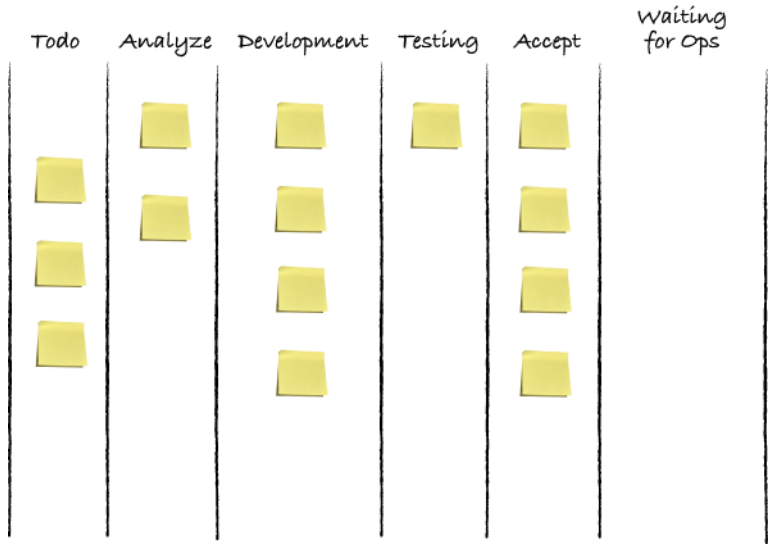
"And the first thing they did was to revoke all our access to the production servers," Daphne muttered.

"I take it that you have to wait a while before stuff is shipped, then?" Joakim asked.

"Yeah, they do deployments six times a year. And they're not going to change their minds." Eric was disappointed.

"Write 'Waiting for Ops' or something," Adam said.

"Yes, let's go with that," Frank said.



"But it could be called Done or Out of Our Hands, if you like," Frank said.


Frank

"Why?" Marcus asked.

"It's only Ops work that's left, and our hands are tied," Daphne said.

Frank continued, "Yes, we don't do much after that."

Joakim and Marcus exchanged a "Should I do it or should you?" look. It was Joakim's turn, apparently, and he asked, "Who is your customer?"

"What do you mean, 'customer'?" Frank asked.

"Well, who is interested in your work or the features you create?" Joakim explained.

"Well, the stakeholders who wrote the requirements for the system—," Frank began.

"Really?" Joakim said sneakily.

"Hmmm … define 'customer' again for me," Beth said thoughtfully.

When the team gave it some more thought, they soon agreed that the users of the internet bank were the important customers to keep in mind.

"'Done' for them would be when they can use the product," Cesar said.

"What about the Waiting for Ops column, then?" Joakim pressed. "Will that make your end users happier?"

"No, of course not," Beth said. "But that's apparently how things are done, so what can we do about it?"

"If we were to ask you to post all the items that were Waiting for Ops, how many would that be?" Joakim asked the whole group.

"A truckload of them," Eric answered before anyone else had time to say anything.

"Yeah—it's many. The last release was—" Frank stopped and thought for a while. "I think there's one coming up this Wednesday. I'd guess maybe 25 to 40 items."

Marcus faced Cesar and asked, "What good are the projects doing your customers there, in Waiting for Ops?" He pointed toward the column.

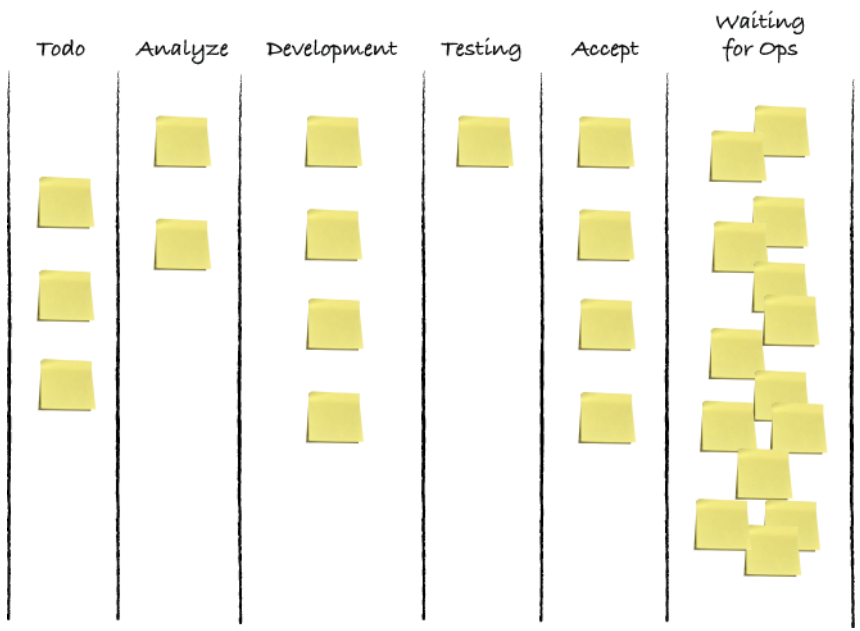
"None whatsoever—we want it out," Cesar said. "But as Beth said, it's pretty much out of our hands."

"Can you change that?" Joakim asked.

"Not easily." Cesar sighed. "They're on a different budget, in a different department, and even in a different building, for crying out loud."

Daphne went up to the board and said, "But if we had a Waiting for Ops column with stickies, and the only thing that prevented those items from getting done were the Ops guys. Don't you think we can start a discussion, at least?"

"Yeah, it's much better than saying 'You suck,'" Frank said, looking at Eric, who in turn looked down at the table.



"We might even be able to help them with some of the simple things ourselves." Daphne saw her chance to get back the admin rights to the servers.

"Yes, it's at least worth a shot," Cesar admitted.

"Great work, team," Marcus said. "When you visualize your pain and gather data about it, it's much easier to get the stakeholders' and other teams' understanding. It's not you nagging, it's data." Eric nodded in agreement and looked relieved.

"Here's another excellent example of why visualization is so important," Marcus cut in. "These issues exist right now; you just didn't see them. By making this simple

visualization, a sticky for each work item and which stage of the workflow it's in, you were able to see a problem in your process."

"Right—even though I'd rather call it an 'unrealized improvement opportunity,'" Joakim said, smiling at Marcus. "And this is a big thing. Your kanban board will tell you all sorts of things. Items moving slowly, items being blocked, or items not being worked on are only a few examples. They're improvement opportunities from which you can learn how to improve your way of working. Seize these opportunities. Call Ops and do something about this, for example. Maybe they don't even know that this is a problem for you!" Joakim realized he was getting carried away and tried to calm down.

"I sure will," Cesar muttered, but he was also quite happy with the progress they had made so far.

"OK, what happens after that—after Waiting for Ops?"

"Well, then it's over," Adam said. "It's in production. Bugs might turn up, but then they're entered as JIRA tickets, hopefully, and that would go in the Todo column, I guess?"

"If you say so, it's your process," Joakim said. "That leaves us with a final column called Done, or something?"

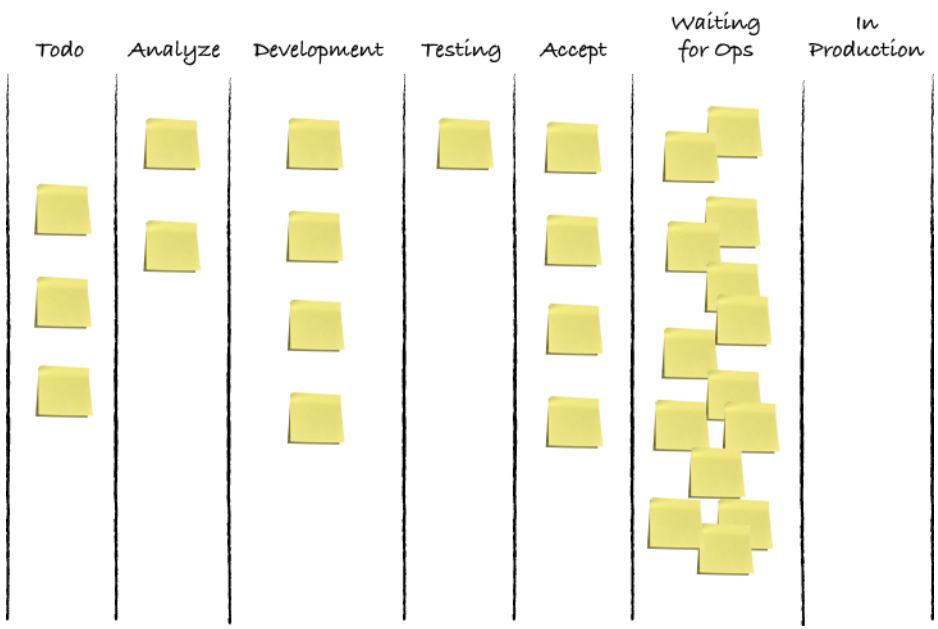
"Why have a Done column at all?" Daphne asked.

"Why not?" Eric replied. "We want to show that we're amazing, right?"

Marcus agreed. "Yes, by all means do." He paused briefly. "And it's also a great way to collect all the work on the board as it's finished."

"Right." Adam got that. "But why Done? I said that bugs might occur. How about In Production? That says more what it is, I think."

The rest of the team agreed, and there were no tickets to put there at the moment.



"Looking at this—would you say that it is roughly the workflow your work goes through?" Joakim asked. "Can you see the status of each item more clearly now than when it was locked in the electronic system?" He nodded to the flipchart with their list of challenges.

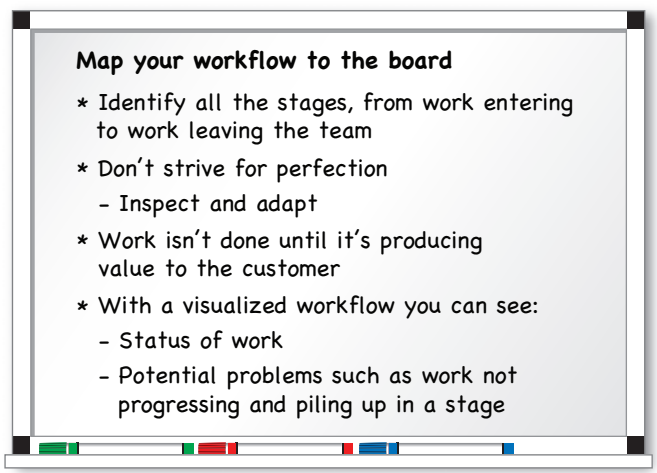
"How about prioritization? That was also one of your challenges," Joakim said, and pointed to the item on the flipchart. "Is prioritization easier for you to see now, when the items are listed in priority order in each column, for example?"

* We often deliver late
* Estimates are often inaccurate
* Team is swamped with work
* Priorities are unclear
* Work is coming to the team from everywhere
* Unclear who's working on what

The team looked at the board for a moment. Frank mentioned that it felt a bit simplified and that they couldn't know if it worked until they had tried it.

"That's alright for now," Marcus said. "Two things: first, this isn't final. This is 'best so far.' This board, and everything else we've discussed, will probably be subject to change later on. We should improve it as we go." He paused for a second, trying to avoid going into preaching mode about the virtues of inspect and adapt and continuous improvement.

"Second, let's talk about the different types of work you mentioned. Or in other words—what goes on the board?"



1.4 Work items

"We now have a lot of stickies on the board, each of them symbolizing a work item," Marcus said. "They each have a short title describing what work item they refer to."

"And the status of each item is clearly shown by which column it's in," Joakim added, sweeping his hand across the board from right to left.

"Is this enough information for all of you, and for your stakeholders, to understand what you're working on at the moment?" Marcus asked.


Beth

"Well, those titles might be too brief at times," Beth said.

"How do you mean?" Marcus asked.

"I'd say that some of them could only be understood by the developers who are working on them, if even by them."

"Maybe you want to write a short description of the work the sticky represents? Not something long-winded, but at the same time not so short that you don't remember what that note was about," Marcus suggested.

"Like a user story, then?" Beth smiled.

"Yeah—you could do that," Joakim answered, "but you don't have to. Any short description that reminds you of what you're doing will do. It's good if it's apparent what the item is about without having to get up close to the board and read for a while. It should be apparent at a glance so you can tell different items apart." Joakim drew a big square on the board and wrote *Description* in the middle of it.

"Is that enough then?" Marcus asked.

"Not always. I often want to know the JIRA number." Eric loved that tool; you could hear it in his voice.

"Why?" Joakim asked, in part because he was not as impressed with JIRA, in part because he thought Eric probably had a point.

"If we're going to have such a short description, we'll need a reference back to JIRA so we can see the rest of the information there. Sometimes there's loads of useful stuff in there. Long discussions, pictures, or what have you," Eric explained.

"Great!" Joakim wrote *JIRA #* in the top-left corner of the square he had drawn on the board.

"Anything else?" he asked.

"Deadlines!" Frank stood up and almost screamed.

"Sure … that seems important to you?" Marcus questioned. He looked at the group: no answer. After a while, Cesar broke the silence. "Yes, they're important to us, but they aren't present on every work item we do. How do we handle that?" He aimed the question at Joakim.

"What do you all think?" Joakim asked the group.

"Let's write them on the notes that have a deadline; it's as simple as that, right?" Eric said from his seated position.

"Yeah, but we want it to stand out, so that we don't miss it." Frank was talking to himself a bit.

"How about using a different color, then?" Beth suggested.

"Great idea!" Joakim went up to the board and added a deadline in the top-right corner. "Another thing that might be useful is to always write them in the same area of the sticky. That makes a connection that the top-right corner has a deadline, if present."

"Great work, team," Marcus added. "But there's one other thing that can prove useful to add to the work item," Joakim said. "Any guesses?"

No answers.

"How do you know who's working on what?" he asked.

That fact was noted on each item in JIRA, but they quickly realized that they needed to move that information over to the board as well. The first suggestion was to write the name, but that would use up space on the sticky; and sometimes more and more names were added, which would make the sticky hard to read.

"A simple thing would be to have some sort of marker to attach to the work item. A magnet with your name or something," Beth said.

"Right: a simple and common solution is to create what are known as *avatars* of yourselves that you put on the work items you work with," Marcus stated.

"An avatar? A blue guy riding a flying horse? How's that going to help?" Daphne looked quite surprised.

"No, not that kind of avatar." Marcus laughed. "A little picture, or placeholder, that represents you. Like a printed picture of yourself, or a cartoon that looks like you, or whatever rocks your boat."


Frank

"Cartoons? But we look nothing like that!" Frank objected.

"Use something that at least resembles yourself, or make sure to add your name to it, so it's easy to make the connection between the picture and you," Joakim said. He was still burned by the time he chose to use stickers with dogs that were impossible to connect to the right people. He started drawing a sketch of an avatar on the board. His drawing skills caused some giggling.

"You know what would be helpful?" Adam said when the sketch was done. "To know if something is a bug or not."

"How come?" Joakim asked.

Adam explained that bugs were prioritized over other work and probably could skip steps like Analyze in the workflow. They also wanted to track which JIRA ticket the bug originated from. After some discussion, the team decided to indicate that

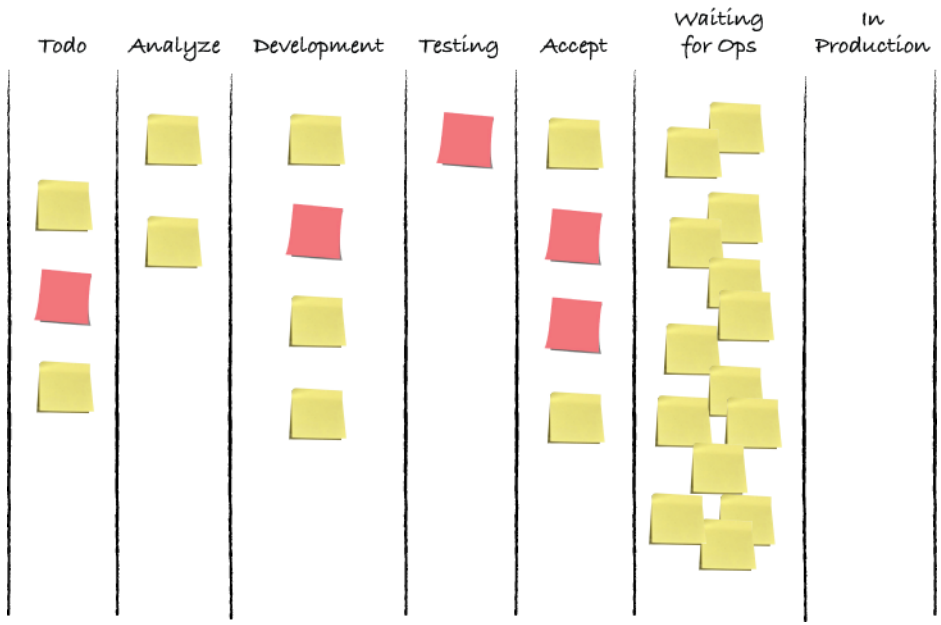
something was a bug by using a different color, writing it on a red sticky instead of a normal yellow one (a pattern common in the kanban community).

"That's really good!" Beth, who always thought of herself as a designer in disguise, was intrigued. "Then those bug items will be easy to pick out even from a distance."

Joakim produced a red sticky and wrote the word *Bug* on it. On a yellow sticky, he wrote *Normal*. He posted them on the side of the board.

"Please come up to the board and change the colors of the items already sitting there," he said, handing out red stickies.

The group discussed some items, but for the most part they were quick to decide the type of work for a particular work item. After a short while, they took a step back and looked at the board.



"Now we can see a little more about what we're working on," Marcus said. "Furthermore, if the board starts to be only red stickies, we need to sit down and have a serious quality talk, right? The board is beginning to speak to us, sending us information about the status of our work. Can you see what we meant by *information radiator* before?"

There was nodding all around. It was easy to see the power of visualization and how simple measures, like changing the color of a sticky, could be useful to help important features to stand out, such as the type of work represented by the note.

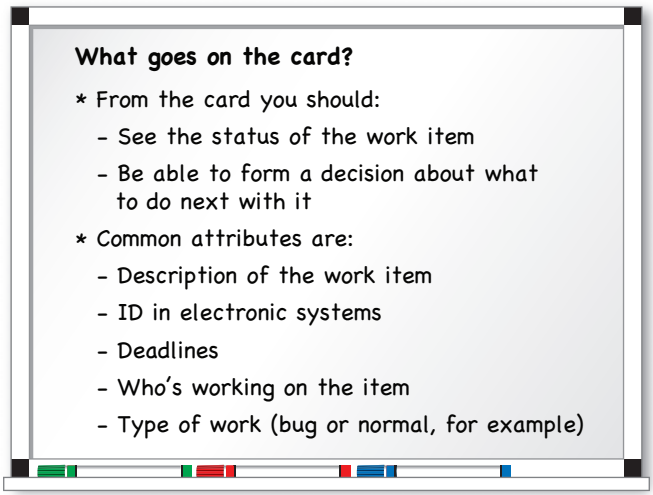
"Great!" Marcus clapped his hands together. "We now have a board with a workflow and a simple template for what should go on each work item. Let's see how work items should move across the board and what we can learn from that."

"Remember where we started on this visualization journey," Joakim interrupted. "We want the board and everything on it to *radiate* information to us. That information will tell us how our work works so that we can learn from it. The real gain isn't seeing the status of each work item, great as that is. The real gain is to help us make decisions and to improve our process as we learn from how it works. This is only the beginning, and you're never done."

"This is a basic and important principle in kanban: to make work visible," Marcus cut in.

"Right, and the basic reasoning behind that is that you can't improve what you don't see. Do you agree?" Joakim said.

REMEMBER You can't improve what you don't see.



People nodded their heads, seemingly in agreement. "Totally," Cesar said, "and already we've seen the value of this. But what do we do next?"

1.5 Pass the Pennies

"Let's check in with your challenges again," Marcus suggested. "Let's see if we can do anything to overcome your feeling of being swamped with work, shall we?"

"That will take us to another important kanban principle, which teaches us to *limit* the work in process," Joakim started. "That is, to strive to work with fewer items at the same time."

* We often deliver late
* Estimates are often inaccurate
* Team is swamped with work •
* Priorities are unclear
* Work is coming to the team from everywhere
* Unclear who's working on what •

He almost didn't have time to end the sentence before Adam, ever the skeptic, said, "Why? Shouldn't we do as much as possible? Right now we're showing the work we're doing; we can't do much about that. We're going as fast as we can."

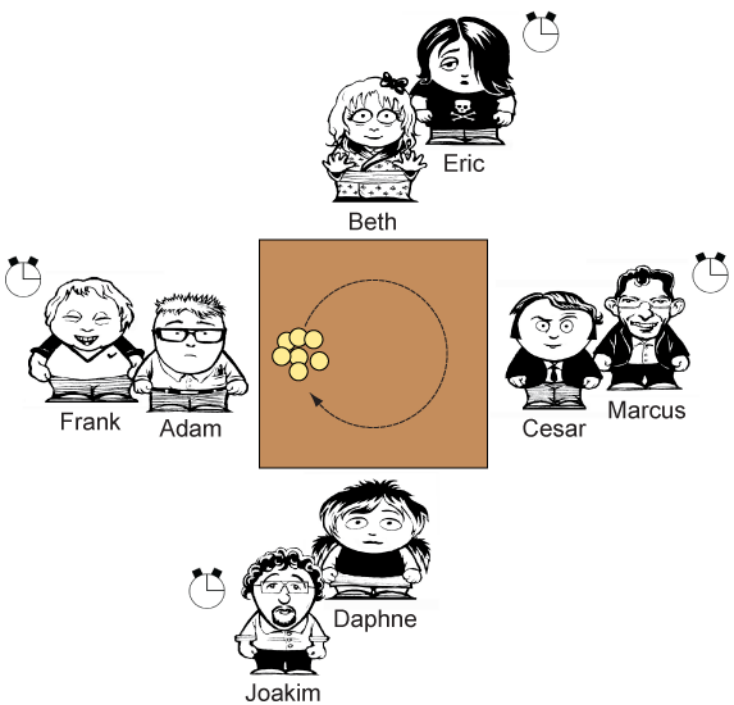
"I don't doubt you for a second. But I think you can do better," Joakim said, a mysterious smile on his face. He looked at Marcus, winked, and said, "I think we need some pennies over here." He pulled out a table and four chairs.

"Right." Marcus produced a small coin purse. He emptied it, creating a little pile of 20 coins at one end of on the table.

He turned to Cesar. "How are we doing with time?"

"We're 45 minutes in. An hour and 15 minutes to go. Please go on," Cesar said mechanically, wondering to himself how these pennies would move things forward.

"Four of you please take seats around this table." Joakim invited the group to the chairs. "The rest of you each stand behind one of your colleagues and bring up a timer on your phone. Marcus and I will fill the last two spaces behind you."



He explained that the aim of the game is for each station to flip all coins once and then pass them on. When all workers have flipped all coins, they're done and delivered to the customer. Everyone seemed to be with him so far.

"The people standing behind a 'worker,'" Joakim made air-quotes to emphasize the irony, "are responsible for timing the worker in front of them to see how much time is spent on the task. Start the timer when your worker *starts* working, and stop it when the worker *stops*. Got it? You're timing the entire time your worker is active."

"Finally, we want to measure the total *lead time*, the time it takes from start to finish for the work to flow through our workflow," Joakim said. "I'll act as the customer and will be timing the whole chain. I'll track two things: the time it takes until the first coin arrives to me, the customer, and the total time—that is, when all of the 20 coins have arrived to me."

After a moment of razor-sharp thinking, Daphne spoke. "But working like this, it means that the first coin and the last coin will arrive at the same time?"

"Yes, that is correct," Marcus responded. "In this first iteration at least. We'll see changes to that soon, in the iterations to follow."

Marcus concluded the introduction to the exercise. "We're going to run this three times. The first time every worker will flip all 20 coins and then send them to the next worker in line. Are you with me?"

"Are you ready? Timers, time your worker's effective working time," Joakim instructed a final time. "Ready, set, go!"

There was intense silence as Adam started to flip his coins like a madman. He passed his coins over to Beth and turned to his "manager" Frank behind his back. "Done—stop the watch!"

Beth continued with equal concentration and sent the coins down to Cesar. Cesar didn't even breathe during his turn and drew one big breath as the coins were sent to Daphne in the last station. Daphne tried to flip two coins at once, one with each hand, resulting in even slower progress. The others moaned. "Come on Daphne—don't let us down here!"

	20		
Adam	10 s		
Beth	12 s		
Cesar	11 s		
Daphne	17 s		
First	50 s		
Total	50 s		

Finally she was done and sent the coins to Joakim, who stopped his watch. On a flipchart, Marcus created a small table and wrote down the results.

Joakim went around to each of the "managers" and asked them for the individual times for each worker. After adding up the total, he turned to the group. "As Daphne predicted, the first and last coin came in at the same time."

"Now let's try with five-coin batches," Marcus instructed. "That means—flip five coins and then pass them down to the next worker before you move on to flipping the next batch of five, and so on. Got it, workers?"

The workers nodded and started to focus on the coins again.

"And managers, make sure to time the *whole* time your respective worker is flipping coins. Starting when the first coin comes in and is being flipped and ending when the last coin has been passed over to the next worker."

"Hmmm—I think I know where this is going," Frank said to himself.

"Time waits for no one," Joakim interrupted. "Ready, set, go!"

This time there were more sounds from the coins but still a sharp focus and silence from everyone in the group. When Adam had flipped his last coin, he cheered for the others. "Come on! Put your back into it, Cesar!"

It wasn't long before the five-coin iteration was over and the results added.

	20	5	
Adam	10 s	14 s	
Beth	12 s	17 s	
Cesar	11 s	13 s	
Daphne	17 s	15 s	
First	50 s	18 s	
Total	50 s	34 s	

"What? That doesn't look right," Cesar called out as he looked at the numbers. "You sure you timed that correctly?" he asked, looking first at Marcus and then at Joakim.

"I assure you it's correct. Let's discuss this further after the next iteration," Marcus responded. "This time we'll do a single coin at the time. Flip it and pass it down. Got it, workers?"

"As before, managers, time the *whole* time your worker works. Ready, set, go!"

Total silence, except for the continuous sound of coins sliding around the table. After what felt like a short time, they were done. Joakim collected the times for each worker and added them to the flipchart.

	20	5	1
Adam	10 s	14 s	18 s
Beth	12 s	17 s	19 s
Cesar	11 s	13 s	15 s
Daphne	17 s	15 s	15 s
First	50 s	18 s	4 s
Total	50 s	34 s	20 s

"I can't believe that!" Frank sounded skeptical and almost as though he felt cheated. "How on Earth can that be right?" The others also looked quite surprised.

"It's correct, all right. Let's discuss what we can learn from this," Joakim said.

"First, take a look at the time for the first coin to arrive to the customer. What happens with that as we switch to smaller batches?" Marcus asked rhetorically and pointed to the summarized rows.

"It goes down, way down! Fifty versus four; that's amazing!" Frank shook his head in disbelief.

"And what about the total time?"

"Well, that went down too, of course," Daphne said, not as impressed.

"But not as dramatically as the time for the first delivery," Frank still could not believe that number. He looked at it over and over.

"And now—take a look at the times for the individual workers," Joakim said, giving them a few seconds to reflect.

"What's the common trend? Not pointing fingers to anyone in particular." He coughed and winked at Daphne. She got the joke and waved him off.

"Well, as strange as it sounds, those times go up at the same time as the others go down. How can that be?" Cesar looked to Marcus and Joakim for answers.

"What we've shown you with this simple exercise is that when you decrease the number of concurrent or simultaneous ongoing work items, the *lead time* decreases," Marcus explained. "Notice that we're doing the same amount of work, but working in a different way—with smaller batches, with less *work in process* at the same time."

"Or WIP, as you'll hear it referred to in the kanban community," Joakim added.

REMEMBER Work in process (WIP) is the number of work items you have going at the same time. Less work in process leads to quicker flow through your process: shorter lead time.

"It wouldn't be much of a community if it didn't have its own strange three-letter acronyms, now would it?" Marcus said, with a big grin on his face.

"As you've seen, this has a dramatic effect on the total time, which in this case went from 50 to 20 seconds." Joakim pointed to the board. "But another thing also happened. When we worked with big batches of 20 coins, we delivered the first and last coins at the same time; but with smaller items, we got the first one in after 4 seconds. Less than a tenth of the time for the first delivery when doing 20-coin batches."

"Smaller batches," Marcus made a shrinking movement with his arms, "with less work in process, will both give you better total speed and let you become more agile, because you can deliver the small, important stuff first. Do see why this could be an advantage?"

"Not only that," Marcus continued "but what if there was something wrong with the way you flipped the coins? What if the customer expected them to be flipped to stand on the edge—what would have been different with different batch sizes?"

"Because we didn't deliver until all of us were done with our individual contributions, we would have to do it all over again," Daphne said. "In the first iteration, that is."

"Finally, we stopped focusing on using each worker as efficiently as possible," Joakim said, directing his comment to Cesar. "In the first iteration there was a lot of waiting until the last worker did any work, but each worker was efficient and worked through the entire batch before handing it down. On the other hand, in the last iteration, when the lead time was the shortest, every worker worked for a longer time, making them less efficient as individuals but more effective as a team."

REMEMBER Optimizing your process for quicker flow can lead to poorer resource utilization.


Adam

"I hate to rain on your parade, but …" Adam leaned back and crossed his arms over his chest. He let out a big breath. "Our work, the 'reality,' is a bit more complicated than this. I hope you realize that?" Adam sounded tired.

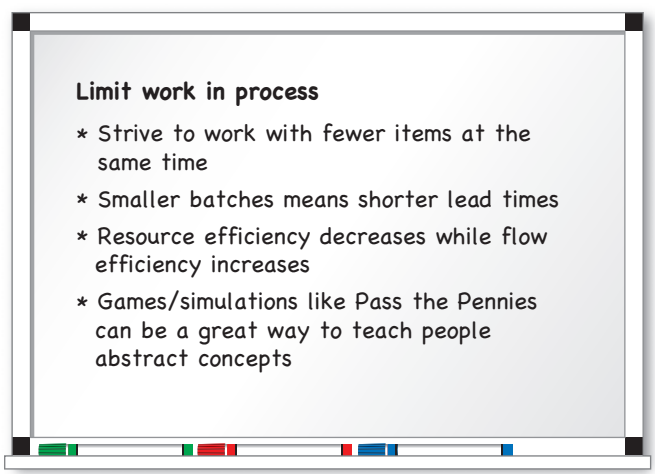
"How do you mean? Please explain." Marcus tried to open a discussion.

"For starters, items don't arrive in a continuous flow like that. They go back and forth between us. I do a little testing, then there's some more coding, more testing again, and maybe even changing requirements.

"Second—the items we handle vary a lot in size. The coins and the work are exactly the same all the time."

"Good points," Joakim interrupted, as he saw Cesar checking his watch. "And you're right. This is a simplification, a simulation that aims to show a single principle: that less work in process leads to lower lead times. In reality, there are a lot of other forces at play, and you have to make trade-offs, but the basic principle still holds."

"There are also a lot of other advantages with shortening lead times and limiting the amount of work in process—but we don't have time for that now." Marcus tried to wrap up the discussion. "This exercise is used as a bit of an eye-opener, and for now the important thing is that you understand the principle. Let's get practical again and gather around the board; how can you apply this to your work and to your board?"



1.6 Work in process

They went back to the whiteboard.

"Where are our pennies?" Adam said dramatically, underlining the fact that he still didn't trust the result fully—not yet, at least.

"What are the pennies on this board?" Joakim asked the group, ignoring Adam's comment.

"They're the stickies," Beth said.

"What about them? What did you learn in the game?" Marcus asked.

"OK, let's see if I understand this," Frank began. "If we want the work to flow fast across the board, we should do fewer items at the same time. Going from 20 coins to 5, so to speak." He looked at Joakim and Marcus.


Joakim

"Do you agree?" Joakim looked at the group. They seemed to agree.

"But how do we do this in practice?" Eric wanted to know. "What do we change?"

"Well, if we all agree that this is a good idea," Daphne replied, glancing briefly in Adam's direction, "that should be enough, right? We agree to work on fewer things at the same time."

"Yes, you could all agree to *stop starting and start finishing*." Joakim mused on this kanban call to arms that he was so fond of, but the blank stares from the audience told him he had to elaborate. "Rather than starting on a new work item, you could help someone on the team finish one already in progress."

"And rather than allowing yourself to be blocked," Marcus added, "for example, by waiting for information or a review from someone, you should try to resolve the situation or work on how to avoid it in the future. These are the unrealized improvement opportunities we talked about earlier, remember?"

"This sounds great in theory," Eric agreed, "but in practice it's not that easy to help someone finish something. I mean, I don't want to bother Daphne by having her explain what she's been doing and how I can help. Isn't it better if she finishes that herself?"

"That's certainly what it's going to feel like at times, especially in the beginning, but you have to start somewhere and use your own judgment," Joakim explained. "In our experience, the path of least resistance is often to start new work, so we have to make a conscious effort to design the work so that it's easier to help each other finish it."

"Which leads us to the next step in limiting work in process," Marcus interrupted, concerned that Joakim was about to go off on a tangent, "which is to use visualization to make the limits explicit and specific. We choose a maximum number of work items that we're allowed to have in progress at the same time."

"This limitation will help us focus on getting stuff finished and help the work to flow faster through the system." Joakim was back on track again. "Limiting your work in process will help you with a couple of the challenges listed at the outset. First, things will start to move along so you'll stop have people breathing down your neck."

Joakim went up to the flip chart and pointed to the first item. "When stuff starts to come out of your system on a regular basis, the demand for accurate estimates and predictions will diminish, in our experience. Second, you won't feel swamped with work because you now have a limit on how many things you'll work on at the same

time. If someone wants to add a new work item, they'll have to also decide what gets taken out."

Joakim added dots to the flipchart as he talked.

"Which, in turn," Marcus cut in, a happy grin on his face, "will make it easier to prioritize. Fewer items to start with, but you now have something to show and reason about. The work you're doing is right there, on the board."

"But how do we know what that number should be?" Beth said.

"It depends." Marcus opened with every consultant's standard answer, realized it, and quickly tried to move on to something a bit more useful. "In general, a lower limit is better, but too low of a limit could have bad consequences. Imagine that you all work on one single item. That would be great for the flow; the second it's ready for development, the developers could pick it up and start working on it. And Adam is getting ready to start testing it when development is done. Any problems with that scenario?"

* We often deliver late
* Estimates are often inaccurate •
* Team is swamped with work •
* Priorities are unclear •
* Work is coming to the team from everywhere
* Unclear who's working on what •

After a moment Frank broke the silence. "Seems like there's a lot of waiting in that case, or am I missing something?"

"No—that's exactly right." Marcus gave Frank a thumbs-up. "Low WIP leads to a lot of slack. Which is good for lead times, but most companies are probably not willing to pay for people sitting idle."

"You need to balance those two against each other: fast flow versus people having work to work on. You want a low work-in-process limit, but probably not a limit of 'one,'" Joakim concluded.

"But that still doesn't help us figure out what that limit should be for us," Daphne said.


Daphne

"No, and I'm afraid we can't tell you." Joakim shrugged. "That is something that you must come up with and constantly tune to increasingly get better and faster flow through your workflow."

"Change the limit of the number of concurrent items. OK—we got that. But from what to what?" Frank was losing patience. "You've got to give us something!"

"Start simple," Joakim said. "Start with one item each on the board. Pretty soon you'll see some problems piling up somewhere, where developers have a hard time keeping up, for example."

"What *ever* do you mean by that?" Daphne said half-jokingly, but there was some sting in there too. She was fast, and she knew it.

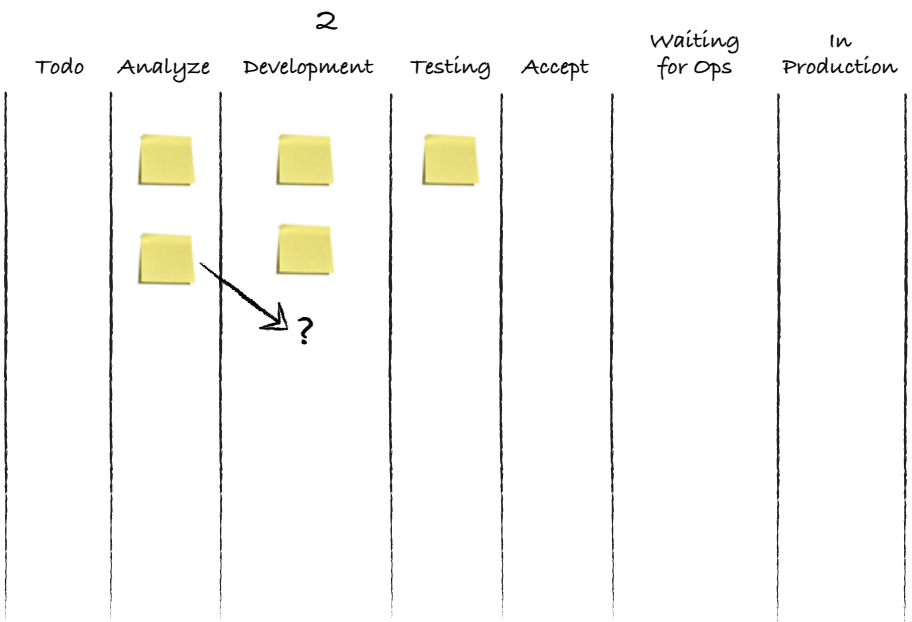
"Some tasks take a longer time to finish than others," Joakim said. "It's not that important where the flow slows down or stops. The important thing is what you do about it."

"Take this example." Marcus cleared the notes from the board. "Let's say we think that two items are a reasonable amount to be developing at the same time, because there are two devs, right?" he checked with Daphne.

"Sure," she answered a bit defensively, cautious where this would lead.

Marcus wrote the number 2 above the Development column.

"We have two items in there in progress. Now the analysts are ready with their work and want to push it into development. That will break our limit of two items, right?" Marcus drew an arrow and a question mark on the board.



"What do we do now?" Joakim asked.

Frank quickly volunteered his opinion: "Put them in there. They'll get there sooner or later anyway."

"Who else thinks that 'putting them in there' is a good idea?" Marcus asked.

"I sure don't!" Daphne retorted. "Wasn't that the whole idea with all this passing of pennies? Keep the number of items going on to a minimum?"

"Right—but what could we do instead, then?"

"Remove other stuff if what's coming in is more important," Eric suggested.

"Do nothing? Wait? Work slower?" Beth pictured the scene, and it didn't sit right with her.

"See? A discussion is ready to take place," Joakim said. "And that's all the work-in-process limit is: a trigger for discussions. Preferably discussions on how to improve

your process so that you can keep the WIP down and have the work items flow faster. It could be a discussion on what specific action to take to avoid bringing more work into the process. Sometimes it will mean a developer helping another developer to finish something, sometimes it will mean replacing something already on the board, sometimes it will mean that developers stop developing to help out with testing—"

"Hold your horses! That will never happen!" Daphne cried; her voice and face told the others she was joking.

"As if I'd ever let you!" Adam was quick to join the joke.

REMEMBER The WIP limit isn't a strict rule; it's a trigger for discussions.

"We often end up doing a lot of testing before a release anyway," Eric chimed in, "so I guess it would make sense to do it all along to avoid the ketchup effect. You know? Nothing, nothing, nothing, and then, all of a sudden, BLAM! Everything comes at once."

"There might even be valid reasons to break the limit from time to time." Marcus tried to get the discussion back on track again. "But if you do it often, you might have to review the limit and see if a higher limit would help the work to flow better. If, on the other hand, you rarely or never reach the limit, you'll never have these useful discussions, and you won't have the tension to improve. Then it's time to lower the limit."

"Now you shouldn't have a problem finding numbers for your limits, right?" Joakim had looked at his watch and realized they had to move along.

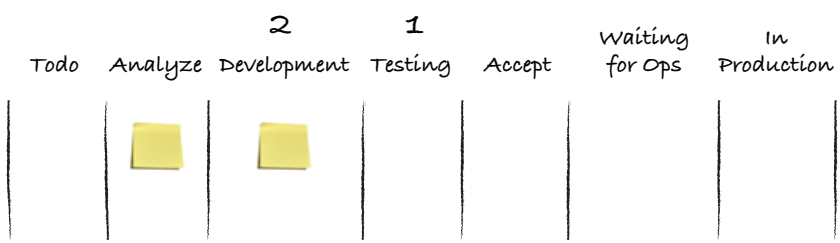
"Well, one each is simple, but is it the correct number?" Frank started.

"Think of it as a limit, and not trying to find the correct one," Marcus said, pressing on. "As we said earlier, you'll inspect and adapt with time. If we decide to go with one item each, how can that be achieved in a simple way?"

"Write the number of people doing work in each column?" Adam suggested.

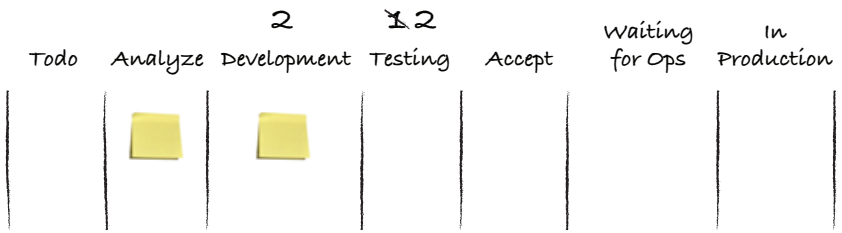
"Yes, great idea! What would that look like? Care to help us out?" Marcus handed a marker to Adam and stepped back.

"Well, I guess, because I'm the only one doing testing right now, we'll end up with a 1 there." Adam wrote a big number 1 above the Testing column.



"Hold on!" Beth stepped up to the board. "I do some testing too from time to time. It's a good way to follow stuff up early, and the analysis work rarely takes up all my time because I often have to wait for meetings to take place. I think 2 is a better number."

"Yeah, that's true. Let's go with 2." Adam changed the number.

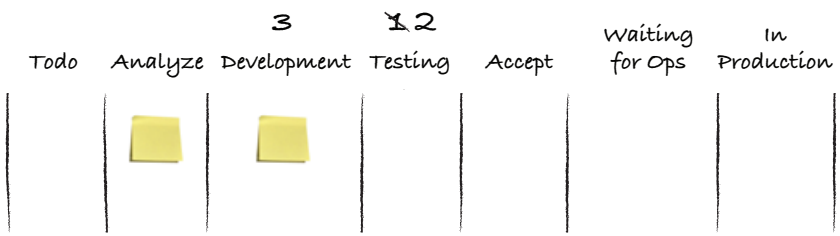


"OK—how about you devs? What's suitable for you?" Adam threw the pen in the lap of Eric, who instantly passed it over to Daphne.

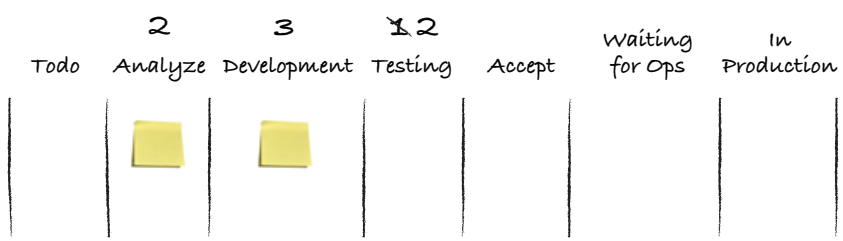
"Two items seems a bit low to me. Sometimes we need to wait for people, which would mean that we would end up with nothing to do. But four seems a lot as well. What do you think, Eric?"

"I say four items; we do loads more than that today. We'll manage!" Eric said with confidence.

"Hmmm—I'll write 3, and we'll keep an eye out for problems," Daphne said as she erased the number that was there already and wrote 3 instead.



"As for analysis work, I think a limit of two items seems reasonable," Beth said, and promptly wrote the number 2 above her column. No one seemed to object to that.



"And the rest of them?" Frank said, turning around. He realized that Marcus and Joakim were nowhere near the board right now. "What should we do with them?"

"What do you mean?" Joakim seemed quite happy to stand in the back and let the team figure it out by themselves.

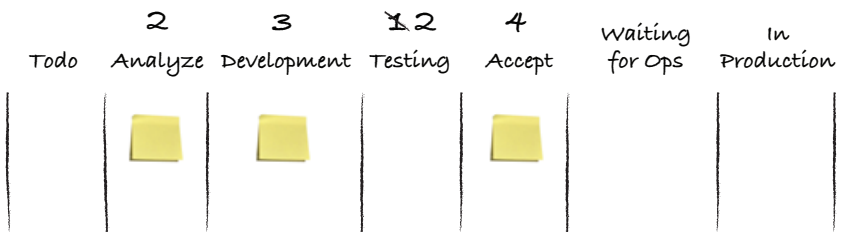
"Well, the Todo, Accept, and Waiting for Ops columns, and so on. Shouldn't they have numbers on them as well?"

"They could," Marcus answered. "But why? What would you gain by that?"

"Eeeh … I thought that all columns should have a number," Frank confessed.

"As we said: they can have them, but there's no rule," Joakim explained. "Put limits on the columns where you feel it helps you get a faster flow. How about that Accept column, for example? What would a limit of items do there?"

"Let's see." Cesar stepped forward and talked slowly to himself. "Limiting that column to, say, four items means if it's filled with four items, then—" He paused and wrote the number 4 above the column.



"When the testing team wants to add yet another one, they will be blocked, and things will back up, so to speak. Because the flow is stopped by that limit." Cesar stepped back. "What does that mean?" he mumbled to himself.

"How do you unblock it?" Marcus offered. The others were silent as Cesar continued to think for himself.

"Well, Beth or I check the items out and accept them …"

"Right—so you could see that limit as a signal to you to come and do your acceptance work." Joakim suggested what he thought Cesar was thinking, but translated into the terms they had been using to introduce kanban.

"OK—but why not have a limit of one, then?" Adam asked.

"Because—" Marcus didn't get further before Cesar interrupted him.

"Because then I'll have to run down there the second something is ready to accept, or the workflow will start backing up." A light bulb was turned on for Cesar. "I see. Very interesting."

"Why four?" Frank asked.

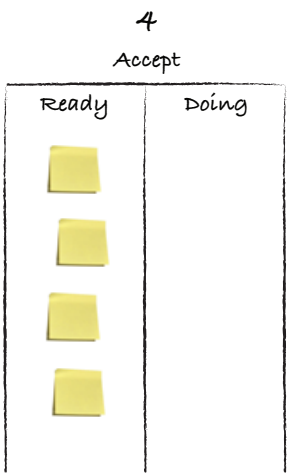
"We don't know what a proper limit is. But four seems like a start that we could use and tweak as we see fit. A lower number demands our constant presence, something we often can't handle because we're traveling or are stuck in meetings. A higher number would mean the team would go too long without feedback from us, which might mean rework if something is wrong; and we want to fix that before they're too deep into something else."

"But why should there be a column for stuff that you're waiting to do?" Daphne asked. "I mean, you're waiting for that column to fill up and then start accepting items when it's filled."

"Good point, Daphne," Joakim answered. "This isn't uncommon at all. It's what we call a *queue* or a *buffer* of work. It's usually put in front of a function that is limited in some way, like the Accept column here. Beth and Cesar can't be with us at all times, so when they get here we want to make sure there's work for them to do. Hence we build a little buffer of work for them, and only call them in when it's filled."

"Not sure I'm following you here, Joakim," Cesar said.

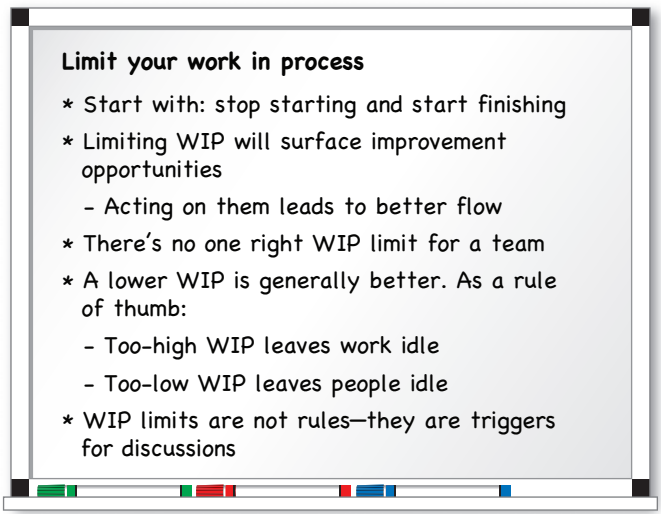
"Let's visualize it, and see if it gets clearer," Marcus said, and went up to the board.



Within the Accept column, Marcus drew a Ready column and a Doing column, dividing the column in two. He turned to the group and said, "Now it's apparent what's waiting and what's in progress. When the Ready column starts to fill up, we call on Beth and Cesar."

"Isn't it like that in Testing and Waiting for Ops as well?" Daphne said.

"It could be. You could have queues where you think they fill a purpose: for example, where there's a limited resource or if you want to visualize the difference between waiting and working. If there's a handoff involved, for example from someone doing development to someone else doing testing, a queue would be a good way of signaling that the work is ready to be pulled to the next stage." Joakim looked at his watch.



Frank nodded; it seemed he was slowly coming to terms with Cesar's explanation. He looked at the board for a moment. "That's it, then? This is the board, we have our limits, and we're done with it?"

1.7 Expedite items

"Yeah … that could be the board for you." Joakim took a step back and crossed his arms over his chest. "What do you think, people? Is everything up there? Is this how you work?"

Marcus went through the challenges on the flip chart, and said: "Even though we might not have solved every point on this list yet, you should now have tools to tackle most of them, agreed?"

There was a moment of silence, and then Adam cleared his throat. "Again—this all looks great, but it's still a simplification," he said, and sighed.

* We often deliver late
* Estimates are often inaccurate •
* Team is swamped with work •
* Priorities are unclear •
* Work is coming to the team from everywhere
* Unclear who's working on what •

"Everything is not new features and moving stuff forward, you know," he continued. "Sometimes things go wrong in production, and then it doesn't matter what you're doing right now or how many things you already have in progress; you fix it! There and then!"

"That's right," Daphne cut in. "Surely we're not holding off urgent work in order not to break the work-in-process limit, are we?"

"Well—you could, but most teams don't, because that would be bad for business. At the end of the day, your bottom-line result trumps keeping your process perfect. How do you treat urgent work today?" Marcus turned to the group. "If you're happily moving along on your new and cool features, and you suddenly receive an alert that the production server is down, what will you do?"

"I'll drop everything and 'run to the hills'!" Eric said, Iron-Maiden-singing the last part. There was giggling all around, because nobody had seen Eric move fast—ever.

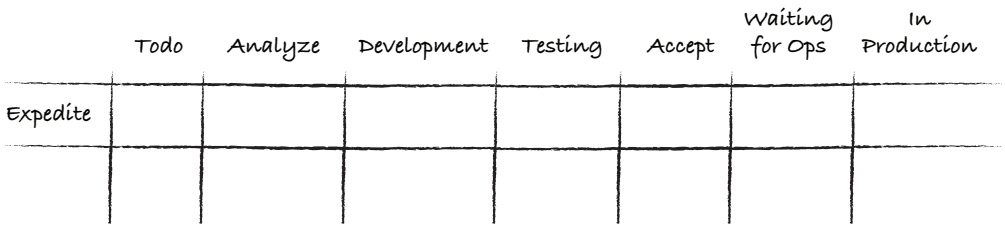
Eric

"Seriously … I'll stop what I'm doing and start to look in to it," Daphne said. "Ah, I get it. We should visualize it on the board. Start where we are, visualize it, and make it explicit, and so on, right?"

"Right!" Marcus gave her a cheesy pistol-shooting sign with his right hand.

Joakim rolled his eyes and quickly moved on. "This is a common scenario. A common solution is to create a special lane on the board for urgent stuff, often referred to as an *expedite lane*."

Marcus, having holstered his imaginary pistol, added a new lane at the top of the board while Joakim explained.



"The policy you hinted at, Daphne, that you should drop your other work and help resolve the issue, is pretty standard; but it's often implicit and not understood or agreed on by everyone. It can be unclear who should act and do something about it or if everyone is supposed to help out. Should the new item be counted against our WIP limit or not? And so on. There are a lot of implicit policies at play."

"Uh-oh, I know where this is going." Eric threw his arms up in the air. "What's the point of sitting here discussing limits if we're going to have this loophole?"

"What do you mean?" Frank asked, giving him an unappreciative stare.

"You guys," Eric nodded his head to Frank, Beth, and Cesar in turn. "You guys could fill this lane with items all the time. It will be your fast lane into the development queue, and you'll use it like crazy, so that *your* work will get done first! Which makes us hop from one item to another. Again. Like last spring with the big Bank 2.0 release, remember?"

There was a moment of awkward silence.

"OK, there seems to have been some not-so-good things going on with expedite work in the past. What could you do avoid that from happening again, then?" Joakim asked.

More pondering. Eric, putting some extra thought into the challenge, fueled by the guilt of having caused the awkward silence, proposed, "Put in a limit of sorts, I guess."

A short discussion followed in which the team agreed on some simple policies for how to handle the Expedite lane. Joakim summed up the discussion: "Let me see if I've captured your policy correctly. An expedite item should only be used for urgent cases and is not to be used as a fast lane to cheat the system. You'll settle on a limit for how many expedite items you allow per month, OK?"

"Well—I for one am happy with that setup. I'll promise you all that we'll handle the expedite items with respect and not cheat the system," Cesar said and held his hand over his heart for the last part. The laughs that followed were a relief from the slight tension that had crept into the room.

"Let's make sure you understand the implications of this policy. What would happen if you put an expedite item into the system?" Marcus asked.

"It gets done faster. That's what we needed," Cesar said, but realized that he might be walking into a trap. *Two can play that game,* he thought, and asked a question back. "Why are you asking?"

"Well, I meant what will happen with all the other work already in your process? Adam, if you think back to the pennies game we played, what would an expedite item look like there?"

Adam thought for a while and then started to reason out loud. "I guess that would be introducing a new coin that had to be passed around before the others." He paused. "And that would not be great for the flow, because we would have to stop flipping the other coins …" All of a sudden, he got it. "That would make all other coins move slower through the system."

Adam

"Correct!" Marcus said. "Use the expedite lane with care. Sure, it will take that item through the system faster, but it will also increase the work in process, and that in turn will slow down all the work already in there."

Joakim thought that sounded a bit ominous. "That is, make sure you know this and take it into account when playing the expedite item card. There are times when it's the right thing to do to get one valuable or important item out there fast. But it has a cost; make sure you know you're paying it. Like a toll, if you like."

"And we'll make sure the lane isn't misused too." Eric looked happy.

"This could also be a way to handle the work that is handed to you in the corridor," Marcus said, and he made a little dot close to that point on the flip chart. "Sure, you could do it, but is it worth the toll you have to pay for it? A discussion can be had with some real data to back it up."

* We often deliver late
* Estimates are often inaccurate •
* Team is swamped with work •
* Priorities are unclear •
* Work is coming to the team from everywhere •
* Unclear who's working on what •

Expedite lane

* Common way to handle special cases
  - Such as work that is urgent
* Often visualized as a separate lane on the board
* Policies around that lane might be:
  - Only one item can be in the lane at the time
  - Max one expedite item per week
  - Don't count the expedite lane against the WIP limit

1.8 Metrics


Cesar

Cesar went to the board, looked at it, and turned around. He was in boss-giving-speech mode now; they could tell.

"I think I speak for everybody when I want to extend a big thank you for this, and—"

"Now wait a minute," Beth cut in. Cesar looked a bit surprised, but then again, this was Beth. She was worth listening to; he knew that from experience.

"Yes?" he asked, encouraging her to go on.


Beth

"I know we only have 20 minutes left, but there's one area that we've left untouched," Beth said. "How do we improve on this?"

"What do you mean?" Frank was surprised. "This will give us better control over our work than we've ever had before. I think it's a fine start!"

"It is! But as I've understood it, this should evolve, right?" She looked at Joakim for support.

"Yes, but …" Joakim responded, "as we talked about when we drew the board, and designed the work items, and so on, everything on the board is subject to change in order to improve as we go—"

"Great!" Beth stopped him short. "How do we know that we're going the right way?"

Joakim said, "How do you know that you're improving when you make a change?"

"Well, if we're doing better than before," Adam said.

"But against what?" Frank responded. This was his territory. "What does 'better' mean? How do we know? We need some metrics to know if we're improving."

"Here we go," Eric and Adam said together, and others joined in with sighs and despondent voices. Joakim and Marcus looked at each other, a bit surprised by the strong reaction.

The team had apparently been subjected to a company-wide effort to introduce key performance indicators (KPIs)[2] that didn't help them at all. Because the KPIs had to be fulfilled, the productivity of the team had been suffering, as well as morale. Before long, the KPIs were abandoned.

"OK, what do you say about skipping that, then?" Joakim tried to steer the discussion back onto a positive track and waited to get their attention. "We're not talking about metrics like that. These are metrics by *you*, for *you*, to help *you* to find areas in which to improve."

"Yeah, exactly," Marcus said. "Track them on the board, and keep them to yourselves if you like. Or at least only show them to the people you choose."

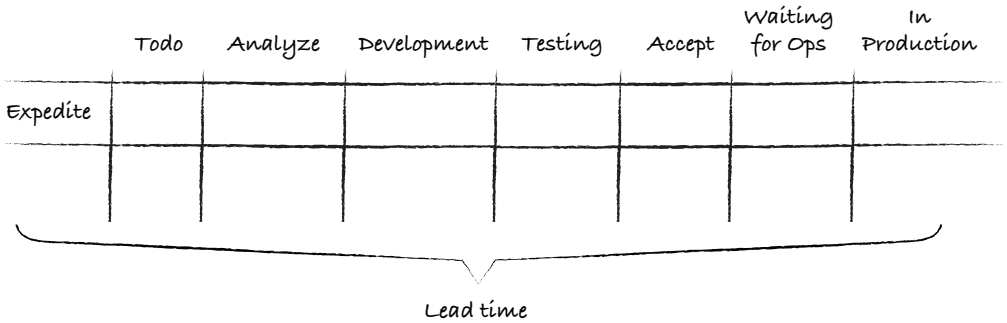
"But it's drudge work to capture and track," Daphne moaned. "I want to do real work!"

---

[2] Key performance indicators (KPIs) are a common way to measure performance.

"It doesn't have to be," Joakim said. "With the board in place, you've set yourselves up to easily track at least two simple and powerful metrics: lead time and throughput."

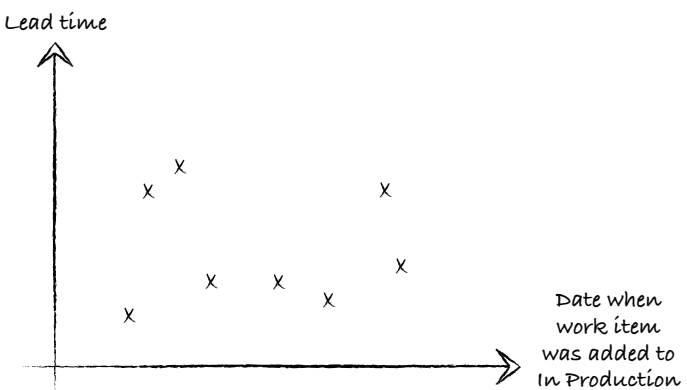
He went up to the board and took a couple of stickies with him. As he spoke, he drew a big brace under the entire board. "*Lead time* is the time it takes for a work item to go from start to finish—from the first column to the last."



"Really?" Adam cut in. "The time our work takes varies greatly. What could we possibly learn from that?"

"Bear with me for a while," Marcus begged. "Starting to track lead times can be as simple as writing down the date of the sticky as it enters the Todo column and then writing down the date when it enters In Production. Plot that out in a simple diagram, and you'll have a pretty good idea of what your average lead time is."

Marcus quickly drew an example sketch chart on the board.



"With a chart like this, you can easily see the differences in time for the different items," Marcus explained. "As always, this is more something to talk about than the solution to your problems."

"But what does that chart help us with?" Adam scratched his head.

"Let's see," Joakim began. "What do you see that stands out to you on that chart?"

They looked at the board, and pretty soon Frank said, "Well, those three." He pointed to the three Xs that scored the highest on lead time. "They seemed to take much longer. Why is that?"

"Why indeed?" Joakim asked. "That might be worth investigating."

"Maybe all of them had to do with a certain subsystem of the application," Marcus offered.

"Or were badly specified," Adam said, looking at Beth.

"Or even lacked testers' input when we did the specification," Beth answered without missing a beat. They both chuckled about it.

"See?" Joakim said. "Another discussion got started. But you could have a discussion even around the general stuff. For example, what would it take to reduce the average lead time by half?"

"Half?" Frank sounded like he thought it was impossible. "It can't be done now. We're waiting so much on Ops and others that—"

"Exactly," Joakim interrupted him, "but what would it take to change that?" He left the question hanging.

Joakim checked his watch and hurried along.

"*Throughput*, the rate at which you complete work, is even easier to track. Count the number of items you finish for a given period of time: for example, every two weeks. Or four, if that suits you better."

"We release the second Wednesday every month," Frank said, "so that would be a great point in time, if you ask me."

Marcus drew an example on the board. "Count the items in the In Production column every second Wednesday, and then remove them from the column. This will give you a simple way to track throughput."

Adam was hesitant. "Again—how will *that* help us improve?"

"Remember before when we talked about lowering work in process to get quicker flow through the workflow?" Joakim asked. "How can you know if you don't track data? Sure, you can go on gut feeling or intuition, but how will you *know*? And will your feeling be enough to convince others if need be?"

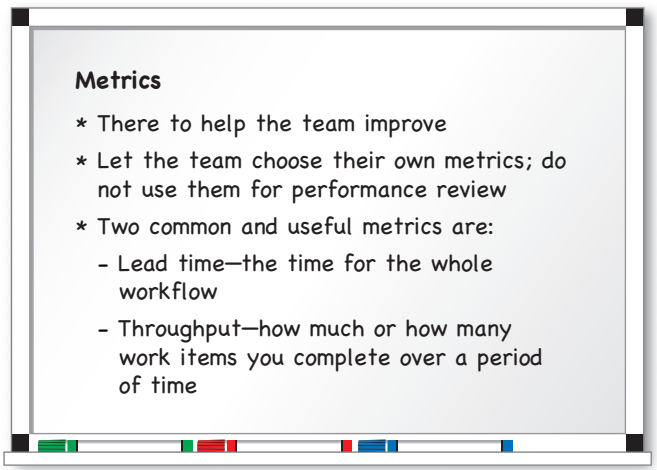
"If you've tracked this metric for a while and average out the result, you could even start doing predictions and make promises you can keep around due dates. Wasn't that one of your challenges?" Joakim continued.

"With these simple metrics," Marcus added, "you can start where you are and track your current process. As you change, you can easily see if you've improved or not."

"But isn't that a bit simplistic?" Adam was still skeptical. "We have some sophisticated systems in place today to track and measure—"


Adam

Eric interrupted. "And we didn't like them and didn't use them, remember? I'd much rather use some simple metrics for us that work and that we use than those stupid ones we worked with because someone in management thought it was a good idea. I distrust most forms of metrics, but these I at least feel I can live with."



1.9 The sendoff

There was silence again. Not as awkward this time, but silence nonetheless. Suddenly, Daphne turned to Cesar. "Time?"

"There's plenty of time—no, wait, we've gone the full two hours right about now," Cesar said, looking at his watch.

"There's much more we could talk about," Joakim said, "but this is a great start that will get you rolling, we think."

"As you progress and want to know more, you can check out the principles behind Lean and try to apply them in your context," Marcus continued. "That can be tricky, so take hints from other teams and how they have applied Lean in their contexts. You can also check out the kanbandev[3] mailing list, the Limited WIP society,[4] and conferences on the topic."

[3] http://groups.yahoo.com/neo/groups/kanbandev.
[4] http://limitedwipsociety.ning.com/.

"With what you have now, you could start tomorrow and continue to improve from there." Joakim tried to close the discussion.


Cesar

"I won't give a speech," Cesar said and stood up. Immediate cheers from the others followed, and when they settled down, he said, "But I want to say: thank you guys. This was a vitamin injection that we needed. And you were right; we could get up and running in under two hours." He smiled.

"When you get home you might want to know more details, some background, and sooner or later a few advanced techniques." Marcus reached over to his bag. "Then this book could help you."

He handed Cesar a draft of their upcoming book, *Kanban in Action*.

"One last question: What should we focus on now?" Frank asked.

"Lead times," Marcus and Joakim said in concert; Joakim continued. "Take the whole process into consideration, and try to shorten the *lead* times."

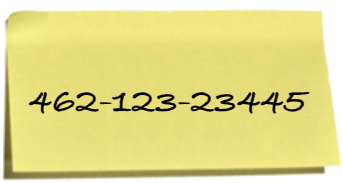
"And here's a hint: try coming up with ways to lower your work in process to get there," Marcus finished.

"Thank you." Cesar looked them both in the eyes and turned away hastily.

At the door, he stopped and turned around.

"Oh that's right—we never talked about your fee," he said, pointing to the table behind Marcus and Joakim.

They both turned around, and at first they didn't see a thing. They approached the table and found a sticky with some sort of number scribbled on it.



"What's this?" Marcus looked back to where Cesar stood a second ago, only to see the door closing behind him.

"Looks like an account number in a bank," Joakim answered. "But what use could we possibly …"

They both got it at the same time.

"We got paid. I think we better get a login to that internet bank, right?" Joakim looked at Marcus, smiling.

1.10 Summary

We leave our heroes there, and the Kanbaneros team is on the way back home, ready to get started with kanban. This whole chapter was a fictional story, and it incorporated some simplifications and adjustments to get the story to flow better, but it has also shown you how you can get up and running easily.

Maybe your work, team, and context aren't exactly the same as for the Kanbaneros, but you can follow along with the reasoning and adjust it for yourself. You don't need to do everything they did; pick and choose what suits your needs best.

This chapter aimed to get you started with kanban. In the next section of the book, we'll examine all the elements described and more in greater detail, offer variants, and warn you about common pitfalls. We'll also offer some more background on where the concepts originated and how they have been applied in other contexts.

You're more than welcome to try kanban out with what you know now, or you can also continue your journey in the next part of the book. In the next chapter, we'll take a short look at the origins of kanban and dive into the principles on which it's built. We'll also give you a cheat sheet for getting up and running in no time.

Part 2

Understanding kanban

After helping you get to know kanban, this part of the book will go into the details of the theories behind kanban. But don't worry—we'll still keep it practical and go into the theory little by little. The focus of this part is the principles that kanban is built around and how those principles can be applied in your team. The topics will range from the theoretical foundations of Lean all the way to how to properly remove a sticky from its pack.

2 Kanban principles

This chapter covers

- Introducing kanban principles
- Getting you started using kanban
- Defining what kanban is

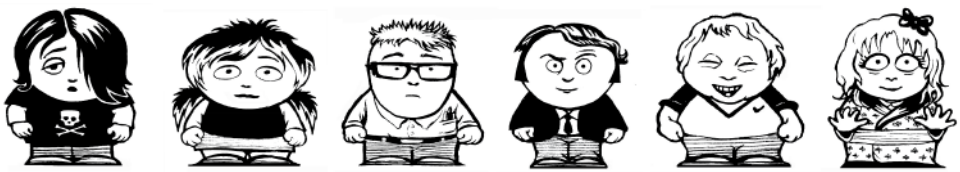
In chapter 1, we introduced you to kanban through a short story. The purpose of this story was to show you, in a practical way, how kanban can be used to help you improve. We hope you already feel that you could use kanban now; but you probably have a few questions, too, such as

- How did Marcus and Joakim come up with all those ideas? Did they pull them out of thin air, or was there some method to the madness?
- That all sounded good, but my team and its situation don't look much like the Kanbaneros; how can I use kanban in *my* team? What do I need to change?
- Was that all? It seemed a little simplistic, didn't it? How can we expand on this?

All of these questions and many more will be addressed throughout the rest of this book. In this chapter, we want to show you the principles that kanban is based on, by looking back on what we did with the Kanbaneros.

But fear not, this is not a lengthy theoretical write-up on everything kanban. If you really want to, you can always go directly to section 2.2 and skip the theory. We'll still keep the theory part short and practical; this is a practical book focusing on practical needs. That's why we'll keep the theory to a minimum and serve it to you in bite-sized chunks as you need it. We'll also give you a quick recipe in this chapter for how you can get up and running with kanban quickly.

You remember the Kanbaneros, don't you? They will pop back into the text throughout the book to ask questions, challenge us, and present practical problems on your behalf. Here they are again:



Had we introduced another process, such as Scrum, XP, or even Rational Unified Process, the first chapter would have looked different. We would have focused on *how* the process should work, what to do and not to do, how long iterations or sprints are, what the tasks are for the Product Owner role, and so on.

Kanban isn't like that. It doesn't prescribe many things at all; it's more like a meta-process that can be applied to whatever process you're working with today. This is great news, because it means you can start where you are today, without changing anything, and improve from there. No new processes, no new roles, and no troublesome reorganizations needed.

Kanban, kanban, or kanban systems?

If you read the text closely, you might spot that we're spelling kanban with a lowercase *k* in this book. The kanban community is continuously improving and evolving, so things change a lot. We have interpreted the current knowledge as follows:

- *The Kanban method* (capital *K*)—A method to create evolutionary change in your organization, first formulated by David J. Anderson and documented in his book *Kanban: Successful Evolutionary Change for Your Technology Business* (Blue Hole Press, 2010, http://amzn.com/B008H1APT0).
- *Kanban* (sometimes lowercase *k*, sometimes capital *K*)—Sometimes refers to a "visual process management system that tells what to produce, when to produce it, and how much to produce" (last time we checked Wikipedia anyway), sometimes to the actual visual signal.
- *Kanban system*—The system that is set up to track the work in process. An example of this might be a kanban board, the cards, and the policies around your work. All of that is your kanban system.

(continued)

In theory, these things have quite different meanings; but in practice, people in the community often don't distinguish between them. What people typically refer to as "using kanban" is to use some sort of kanban system to manage and optimize kanbans (the visual signals) in order to evolutionarily improve. That is what we mean when we use the word *kanban*.

If that was confusing to you, don't worry—you can read the book and make up your own mind about what kanban is, and we promise there won't be a test at the end.

The Japanese connection

You should probably know something about the word *kanban* itself, if for no other reason than so that you can impress your friends with your knowledge of Japanese. *Kanban* is two Japanese words put together: *kan*, meaning visual, and *ban*, meaning card. Put together, it becomes something like "visual card" or "signaling card."

Kanban as a concept comes from Toyota, where it was invented as a scheduling system for just-in-time manufacturing in the Toyota Production System (TPS). When researchers in the West studied TPS, they called it Lean Production System, later Lean Manufacturing and Lean Thinking.[a] Kanban has its origins in the principles of TPS and Lean, which is why you'll find a lot of references to these concepts throughout this book and in most other texts about kanban for software development.

a. John F. Krafcik, "Triumph of the lean production system," *Sloan Management Review* 30, no. 1 (1988): 41–52. James P. Womack, Daniel T. Jones, and Daniel Roos, *The Machine That Changed the World* (Scribner, 1990, http://amzn.com/B001D1SRRS).

Although kanban doesn't prescribe many concrete rules or practices, you still can improve greatly by using it. How can that be? At the heart of kanban are a few principles that can guide you on how to use kanban to improve. Let's take a closer look at those principles.

2.1 The principles of kanban

Kanban is based on three simple principles:

When we got down to business with Team Kanbaneros, we started by *visualizing their work*. To start, this can be as simple as creating sticky notes that represent each work item and a visualized workflow in the shape of a board to track each item's current status. This is a

great way to get to know your work, learn how your "work works,"[1] and start seeing improvement opportunities in your workflow.

For many teams that we coach, this creates a big impact right away. Just making information visible that previously was not can help you solve a lot of problems by itself.

There are other underlying aspects of visualization as well, because you're also starting to make implicit policies around the work explicit. That is, everybody might think they know how you work with a feature request; but by showing the workflow on the board in columns, the real process becomes apparent to everyone on the team. By doing this, you make any differences you might have on policies about work more evident. This can lead to a discussion that clarifies the policies you work from, and you can easily capture them on the visual board so that all team members approach the work in the same way.

With Team Kanbaneros, we went through some common ways of visualizing work: the board, the work items, and the expedite lane, for example. There is much more to know about this, and chapters 3 and 4 will take you through it in greater detail.

We continued with the Kanbaneros by playing a game called Pass the Pennies (see chapter 13 for more details on this and other games). This showed the principle of *limiting work in process*. Simply put, the principle states that you deliberately establish a limit for how many items you'll work on at the same time. The first apparent gain from doing this is that with fewer items being worked on at the same time, each item will be done more quickly. Chapter 5 will explain the effects of work in process in depth.

But limiting work in process (WIP) is also important for other, subtler reasons. By establishing a WIP limit, you'll create a little tension in your workflow; this is a Good Thing™, because it will expose problems in your system, or, as we put it to the Kanbaneros, "unrealized improvement opportunities." You can read more about limiting WIP, the reasons for doing so, and how you can visualize these limits in chapter 6.

Limiting WIP will start surfacing improvement opportunities. Flow through the workflow will stall (stickies moving slowly over the board), start to back up (a lot of stickies in certain columns), or stop completely (items waiting). These are all indicators that you can improve your system. What you do to fix these problems will determine whether you improve.

But if you want to improve, you should know what the goal is, and this is where the last principle of kanban comes into play: *manage flow* quickly and without interruptions through the workflow.

This is the start of your journey to continuously improve the workflow. The bad news is that you'll never be done with this task. Your workflow can always be improved; there's always a bottleneck

[1] See John Seddon, *Freedom from Command and Control: A Better Way to Make the Work Work* (Vanguard Consulting Ltd, 2003, http://amzn.com/0954618300).

that slows you down.[2] The good news is that the problems will reveal themselves to you, visually, on the board. Not only that, but often the biggest problem will be revealed first—solve that, and you have made a big improvement to the flow through your workflow.

Team Kanbaneros discovered several improvement opportunities, as you might remember from chapter 1:

- They realized that the way deployment was done was hindering the flow severely. For the Kanbaneros, we could see this on the board with all those stickies in the Waiting for Ops column. You could say they already knew this, but now they have data constantly in their face, demanding action.
- They talked briefly about bugs, which were handled differently than normal items. You could say that bugs are in a different class of work than the other items and maybe should be given precedence over other items in prioritization.
- The Kanbaneros put WIP limits in place to help their work flow smoothly; for example, they queued up four items for acceptance so that Cesar and Beth didn't have to run to the team several times every day. Instead they now have a reasonable amount of work to do, waiting for them when they come by the team.

Help the work to flow faster through the workflow. That doesn't sound so hard, right? You can always find help in chapter 7, so by all means—make it happen!

There are a lot of things you can do to accomplish this. When working with this principle, you can find inspiration in Lean Thinking to help your work flow more smoothly by removing waste in your process. You can also take a look at the Theory of Constraints[3] and identify, exploit, and alleviate the bottlenecks in your system. Practices from the agile software development community movement might help you to improve collaboration and quality and thereby improve the flow of your system. It's up to you which route you take to improve your system. The important thing is that you react to the signals that your work is sending you and improve on it.

In reality, you'll see the principles combined with each other a lot. For example, in order to get a quicker flow, you limit WIP, which is also shown visually on the board.

With a visualized workflow, a limit for the WIP, and a focus on moving work through your workflow, you have set yourself up to easily spot improvement opportunities. How you go about doing that is pretty much up to you, but we won't leave you high and dry. We have packed chapters 3–7 full of tips, practices, patterns, and common solutions to improve the flow in your workflow.

The search for improvements will soon take you outside the boundaries of your own team. In order for you to get a faster flow, you might have to interact with other teams or functions around you in a different manner. A first step toward this is to include teams or functions before and after yours on your board. Or maybe you could

[2] Well, when software—or rather, solutions—materialize instantly when you imagine them, we guess there are no more bottlenecks. It probably won't happen any time soon, though.

[3] Visit http://en.wikipedia.org/wiki/Theory_of_constraints to read about the Theory of Constraints (TOC).

even have a board for all the teams in your department, aggregating the status of the teams there.

It could be the start of a transformation that can ultimately reach the entire company, in an evolutionary way. Soon you'll find yourself teaching kanban to others and being involved in change management on an organization-wide scale. You can read about this in chapter 13.

Three principles? I thought it was five properties. Or was it six practices?

Kanban is fairly young as a methodology used in the software business. There's a vibrant community, and new things are discovered and put into practice continuously. This is something good, and it's very much in line with the principles of Lean and continuous-improvement thinking.

The three basic principles we describe in this section make up the foundation that kanban is based on. Recently, David J. Anderson and others have extended the three basic principles to five properties and later six practices; these are now referred to as the *core practices*.[a] They are as follows:

1. *Visualize*—Described earlier.
2. *Limit work in process*—Described earlier.
3. *Manage flow*—Described earlier.
4. *Make process policies explicit*—With explicit policies, you can start to have discussions around your process that are based on objective data instead of on what you think, feel, and have anecdotal evidence for.
5. *Implement feedback loops*—This practice puts a focus on getting feedback from your process: for example, in what is called an *operations review*, which is a kind of retrospective for the process itself.
6. *Improve collaboratively, evolve experimentally (using models and the scientific method)*—This practice encourages you to use models such as the Theory of Constraints or Lean to push your team toward further improvements.

That's three more practices added to the principles we've talked about so far. Note that this holds true for the Kanban Method of "incremental, evolutionary change for technology development/operations organizations," and in that context the last three practices are important.

As you probably have noticed already, we take a practical and pragmatic approach, and for us the last three practices fit nicely within the basic principles:

- *Make process policies explicit*—Making policies explicit is what much of the visualization of work is about. As often as not, the important step might not be to write it on the board (although that's important too), but rather to hold discussions that allow you to form consensus about the policy you intend to put in place. Although it's great to make this … umm … explicit, we feel that it's part of the principle of visualization.

a. See http://mng.bz/EkgB.

(continued)

- *Implement feedback loops*—This is part of the "manage flow" step for us. In order to help the work to flow, feedback loops are essential and should be sought and implemented where needed.
- *Improve collaboratively, evolve experimentally (using models and the scientific method)*—We wholeheartedly agree on the importance of this. But this mindset is so deeply rooted in the Lean principles that underlie kanban that we don't think it's a principle of kanban per se, but rather the environment and ecosystem that kanban springs from.

To further complicate things, David J. Anderson and others are now using "principles" to describe some other principles:

- Start where you are.
- Agree to pursue incremental, evolutionary change.
- Initially, respect current roles, responsibilities, and job titles.

And recently a fourth principle was added:

- Leadership at all levels in the organization.

During our time as kanban practitioners, the principles we talk about have become practices, the practices have gone from three to five to six, the term *principles* has been redefined, and a new principle has been added. We expect and hope that the discussion around these practices isn't over. But from this sidebar, you now understand why we're talking about three principles (visualize, limit work in process, and manage flow) when others are talking about five practices. Or was it six?

2.2 Get started right away

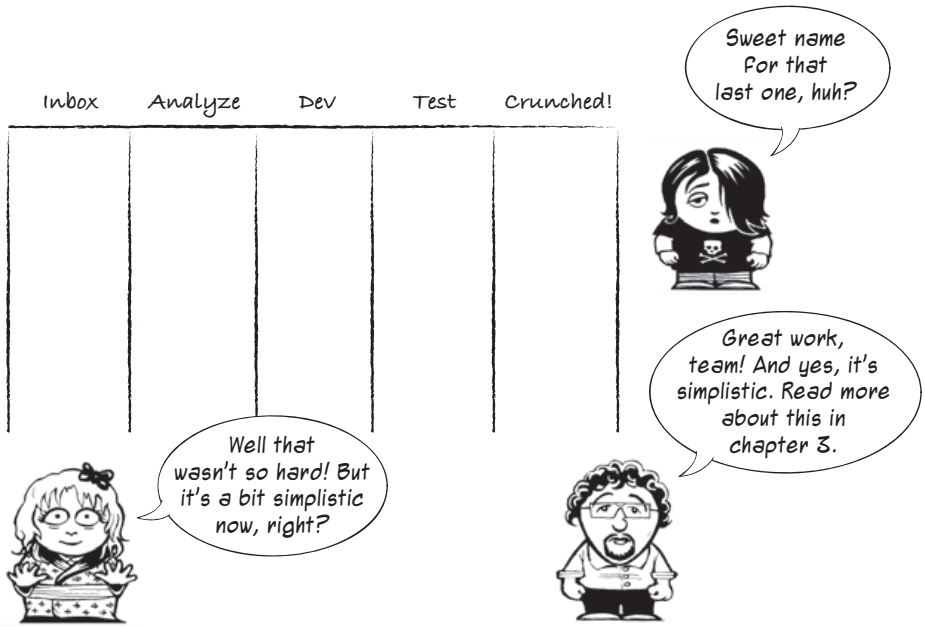
Just like the Kanbaneros, you can easily get started right away, due to the lightweight nature of kanban. In fact—why not start now?

It doesn't take much, and you can start easily right where you are today. Just begin focusing on getting stuff done, really done, before picking up new work. Make that the motto for you and your team: *stop starting; start finishing!* This is a simple way to limit the work in process, but it can be effective.

If you want to be a little more concrete and practical, you can also do an exercise similar to the workshop we ran with the Kanbaneros in chapter 1. Here's a short description of how that can be played out:

1. Start by visualizing your work. We asked the team to create a sticky for each work item they worked on and place it anywhere on a whiteboard.
2. Map your workflow to the board. A simple way to do this is to create a column for each stage in your workflow. Move the tickets that are in the different stages into

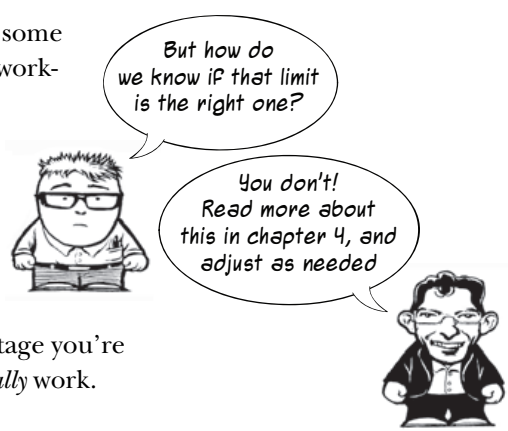
the correct columns. At this stage, the Kanbaneros talked quite a lot about how the work worked, and that's something good that increases your knowledge of the process. But resist the urge to optimize your workflow before you can see what's going on. After this exercise, you might end up with a board like this:



A WORD FROM THE COACH Make sure you draw the board *together* as a *team*. Don't let one person do this by herself—especially not you, if you're an external coach. The buy-in from the team will suffer greatly. Trust us; we've made that mistake far too many times already.

A simple way to make this exercise a team effort is to pass the pen around as you draw the board.

3 Make a couple of dry runs with some work that has passed through your workflow to see that the workflow actually matches the way you work. Change as needed. We didn't have time to do this with the Kanbaneros, but we think it's good practice, and it quickly shows you whether you got the workflow right. Remember, at this stage you're aiming to track down how you *actually* work.



4. Decide on a work-in-process limit—how many work items you, as a team, should be working on at the same time. We spent quite a lot of time here as we helped the Kanbaneros, but please don't overthink it. It's better to get something up there and improve as needed. For now, go with two work items per person in the team (six in our example), for example, and spread them evenly across all columns. Or get some good ideas in chapter 6, which is all about WIP limits.
5. For extra measure, create some avatars—small pictures of yourselves—and attach them to the things you're working on now. This will help you more easily see what's going on and who to turn to if you have questions regarding one of the work items.

There—you now have a simple kanban board to improve on. But even this simple tool will help you to spot problems and continuously improve, in small steps. The rest of the book will help you learn how to do that.

2.3 Summary

Kanban is an approach to software development based on simple but powerful ideas. It aims to make the work *flow* fast through the whole value chain, from idea or concept to software in production, delighting your customers. The tools that make this happen are a few simple yet powerful principles: visualizing your work and policies, limiting the amount of work in progress, and helping the work to flow better through the process.

These tools can guide you to continuously improve your process to gain an even faster, smoother flow of work through that process. This is a never-ending quest that will help you and your team to improve, little by little, every day.

There—now you're up and running. As your work progresses, move the items accordingly on the board. Take care of problems that are hindering your flow. When a problem occurs, stop and talk about it; these are your improvement opportunities! Don't waste them; handle with care!

The rest of this book is packed with practical advice about visualizations, how to limit WIP, and different ways of helping your work to flow smoothly and quickly through your workflow. We have put a strong focus on practical matters to make sure you can use the practices, patterns, and tools we describe right away with your team.

Let's jump right in and start looking at visualization—a subject that will get your creative juices going.

3 Visualizing your work

This chapter covers

- Using visualization for transparency and information sharing
- Making policies explicit
- Using information radiators
- The kanban board
- Mapping your workflow to the kanban board

The first thing that strikes you as you enter the Motomachi Toyota assembly plant is how Toyota uses *visualizations* and sound to track progress, display status, and communicate. The company has large signs everywhere explaining what different parts of the plant are for, and big *Andon boards* that show the status of production, including when and where assistance is needed. When you remove a tool from its place, a bright image of the tool is revealed so that everyone knows the tool is gone from its spot and remembers to put it back.

If a worker encounters a problem, he pulls a cord—the famous *Andon cord* or *stop-the-line* cord—and his foreman and coworkers are alerted by a tune playing, unique to that workstation, and by flashing lights on the Andon board that indicate

which station is in trouble. At the same time, a yellow lamp lights up next to the station; and if the issue isn't resolved until the vehicle on the assembly line passes a marker for the next station, the lamp turns red and the line stops.

Another big board in this plant is shaped like a car seen from the front; one of its "headlights" is a clock. It hangs from the roof, big and visible to all, displaying information such as the planned number of cars to assemble, the actual number assembled, the plan for the current shift, and the percentage of the target achieved. A light indicates whether a quarter or half an hour of overtime is going to be needed to meet today's target.

These are only some of the visualizations used to help the workers act and make decisions; there are plenty of others all over the plant. Visualization is so important to "the Toyota Way" that one of its principles is "Use visual control so no problems are hidden."[1] This seems to be deeply ingrained in Japanese culture. If you have ever been to Japan, you may have noticed that the Japanese use all kinds of nifty visualizations to help in everyday life. When Japanese use the term *visualization*, or *mieruka* in Japanese (**見える化**), they often mean not only presenting things in an easily understandable visual form, but also the goal of greater transparency and information sharing among employees and stakeholders in order to increase the organization's effectiveness. The term also refers to a series of processes that make use of the visualized results, thinking them over and initiating actions in an appropriate manner.

To us, this is the essence of the principle "Visualize" in kanban: to make all necessary information visible when people need it, enabling effective collaboration and improvement through understanding how the work works. To achieve this with kanban, you have to make policies explicit and use information radiators.

If you remember from chapter 1, when we introduced kanban to the Kanbaneros team, we started by visualizing their work. At first, they created a sticky note for every item they were working on right now. We have seen many teams experience "Aha!" moments from visualizing work like that. Questions like "Are you working on that? Me too. Maybe we should cooperate!" are quickly brought to the surface.

This chapter will show you the best-known visualization tool for kanban practitioners: the kanban board. After reading this chapter, you'll be able to set up and start using a kanban board that is customized to your way of working.

---

[1] Jeffrey Liker, *The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer* (McGraw Hill, 2003, http://amzn.com/0071392319).

3.1 Making policies explicit

Often, when people are working together, a lot of implicit assumptions and policies are at play. People make assumptions about everything from how people are supposed to clean (or not) the toilets after using them to what it means to take responsibility for finishing an important piece of work. Sometimes these assumptions are inconsistent or even conflicting, which can lead to misunderstandings and ineffective teamwork, emotional and subjective discussions of problems, and so on.

If you can make these policies explicit—for example, through visualizing your workflow on a kanban board and talking about what the different steps mean—the inconsistencies have to be dealt with and the conflicts can be resolved. Once the policies are made clear, it's much easier to hold a rational and empirical discussion about how to improve them. This provides a good foundation for collaborative process improvements.

> ... the code is more what you'd call "guidelines" than actual rules.
>
> Captain Barbossa in
> *Pirates of the Caribbean*

Remember that the policies are only that: policies—not rules that *must* be followed. They can, and should, be broken from time to time, but the decision to do so should be made intentionally and often with careful consideration from the whole team.

The Kanbaneros in chapter 1 made a lot of their policies explicit by mapping out their workflow as columns on the board. If you remember, they had a long discussion about how their work behaved in different situations. There was a lot going on in those discussions under the surface, because the team had not formalized the way they worked. In the process of doing that, they got insight into what worked and what didn't.

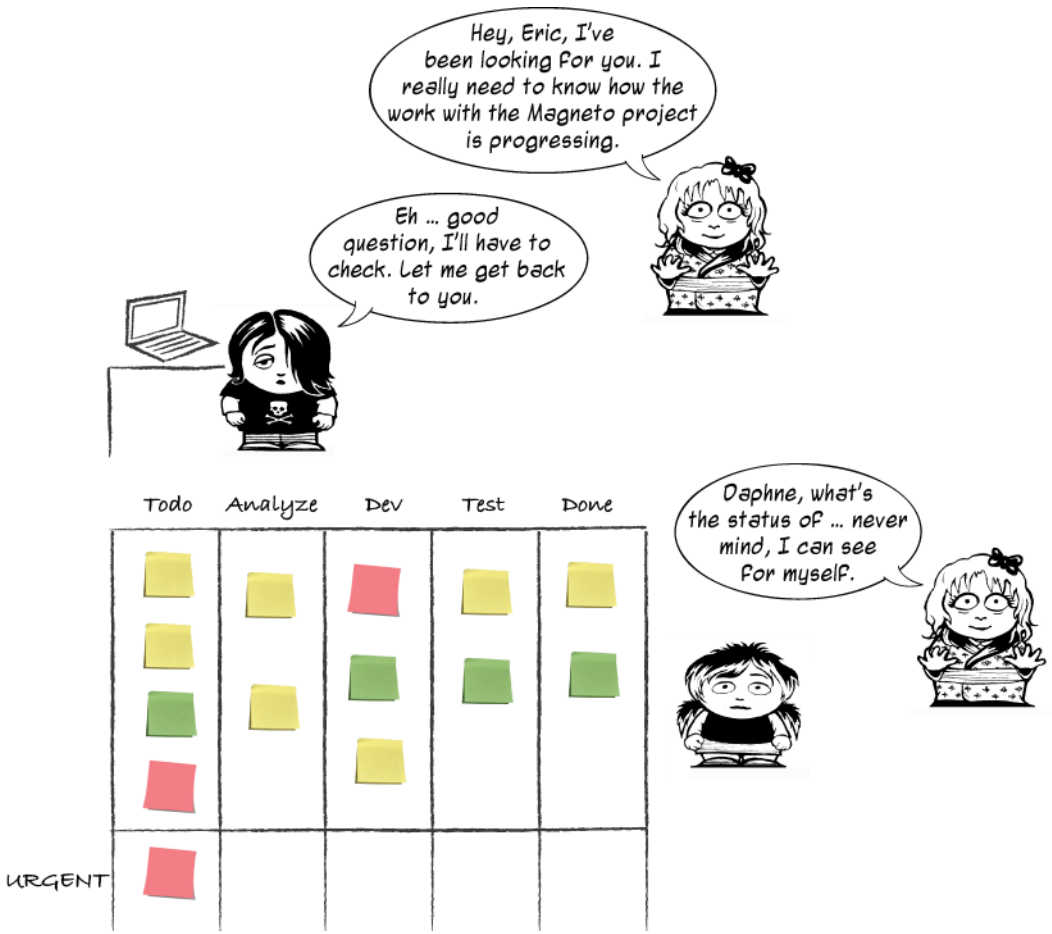
Other policies that the Kanbaneros made explicit concerned the types (colors) of their work items and how they should be handled, the limits for how many work items they could work with in certain columns, and the expedite lane (an example of a so-called *swim lane*).

WARNING Sometimes transparency can threaten people. If you have grown accustomed to doing things your own way—maybe you prioritized work in your own way, did favors for "your people," and managed to fly under the radar for a long time—making policies explicit may not seem like a good thing to you. Introducing an open and visual way of working can seem threatening. Handle this problem by introducing the visualization in small steps, and let the team decide what to visualize and how.

To us, visualizing the work and making policies explicit are inseparable; but because visualization can be more narrowly interpreted, the official kanban definition has

"Make process policies explicit" as one of its six core practices, along with "Visualize." As you'll see throughout the book, a lot of the kanban practices and patterns are about making policies explicit and using visualizations to do it; the most important and obvious practice is the kanban board itself, an excellent information radiator.

3.1.1 Information radiator



Effective teams have to constantly respond to events together and coordinate their activities. In a fast-paced environment, traditional status meetings often feel dry and dreary, and traditional project documentation feels like unnecessary overhead, inaccurate and incomplete. You want information to be up to date and easily available to all interested parties at the time when it's needed.

> *An information radiator displays information in a place where passersby can see it. With information radiators, the passersby don't need to ask questions; the information simply hits them as they pass.*
>
> —Alistair Cockburn[2]

Information radiators typically take the shape of big and visible displays or charts that you can understand at a glance. They can be big posters with charts showing the progress of a project, walls of index cards detailing the actions from a workshop, or whiteboards with columns showing a workflow, with stickies representing the work items. They're placed so that the team can always see them, but they should also be easily accessible to stakeholders outside the team.

The information radiator should be easy to keep up to date, so that it's constantly updated with the necessary information and worth visiting. This is why many teams prefer hand-drawn charts and low-tech equipment such as whiteboards, papers, stickies, and index cards. Perhaps more importantly, this also makes it easy to continuously experiment with and improve the information radiator to make it fit your particular context.

There are a lot of electronic tools that mimic boards and card walls and that are invaluable to teams who might not sit together or who for other reasons have chosen to use them over a physical tool. Digital tools can be great for a lot of reasons, but you run the risk of having the tool deciding, and putting limits on, how you improve your process. Especially when creating a new information radiator, you want to be able to change your visualizations quickly and easily, because it's unlikely that it will be perfect from the start. Choose a technique (electronic or physical) that isn't difficult to change.

The Kanbaneros created a big board with all their work in a sequence of columns. Each of their work items was represented with a sticky note on the board. From these simple and common visualizations, their current status and workload was easy to see and understand, even for someone who had never seen a kanban board before.

Let's take a look at some of the things to think about when you create your information radiator.

DON'T HIDE YOUR INFO IN THE FRIDGE

Digital tools sometimes become information *refrigerators* in which information is hidden. You have to know where to look for it, you have to be able to use the tool, you have to be authorized, it takes effort to go into the tool, and so on. It's the opposite of radiating information; you don't know if the information is even there, and you have to open the door and dig around before you find it.

This is an important point if you have picked a tool that doesn't naturally become an information radiator. A lot of electronic tracking systems today offer a dashboard or screen that looks a lot like a board of stickies. Make sure you display it on a big monitor near the team or even on the wall with a projector. You can read more about tools for kanban teams in appendix B.

---

[2] *Agile Software Development* (Addison-Wesley, 2001, http://amzn.com/0201699699).

Electronic or physical board: pros and cons

There's a lot of debate in teams all over the world about whether a physical or an electronic board should be used. Because there's no one right answer for all, you can only weigh the pros and cons against each other and decide for yourself.

Pros with an electronic board:

- Universally accessible, and hence a great help for teams that not are co-located
- No loss of data, whereas with a physical board, stickies can drop on the floor
- Automatic calculation of metrics
- Can store information and discussions about each item

Pros with a physical board:

- Generally bigger
- Draws people from their desks and becomes a natural place to gather
- Generally easier to set up—you only need a whiteboard or a wall
- Easier to change and adjust to your particular needs

The reason we talk more about physical boards is that they're generally easy to get started with and to change in those early phases. But again, you decide for your team!

CO-LOCATION? LUXURY!

Maybe you and your team don't have the luxury of sitting together. You can try to use a common area or hallway, someplace where you can easily gather around the radiator and where it's easily accessible to others. You could also try to use a digital tool with a projector or a big screen.

Leave it to the team

The decision to use an electronic tool or not should be left to the team. Marcus had the opportunity to do some coaching as a contractor at Spotify, where Joakim works, and found that two out of the three teams he was assigned to used an electronic tool.

That fact in itself was not so surprising, but the team that didn't use an electronic tool was *not* co-located, with a team member in Germany and the rest of the team in Sweden. It was a perfect fit for an electronic tool, but the team felt that using a physical board and sending photos of it once a day was good enough for them.

The other two teams sat not more than 10 meters from each other and still opted for electronic tools, but with big visible monitors that radiated the information to them. That team had strong opinions about using an electronic board, and there was no reason for Marcus to talk them out of that.

There's no right or wrong, only what you think would suit your team best. Make sure your information is visual for everyone in the team to see; that's what visualization is all about.

CAN. NOT. PROCESS.

Avoid information overload. Don't try to fit too many charts, too much info, or too much detail into your information radiator, because this will make your radiator more difficult to understand.

GO BIG

This should go without saying, but experience tells us that it's necessary to point it out at times: make sure the information is big enough to see easily from a few feet away, that text is legible, and so on, so that people can read it without having to get too close and squint.

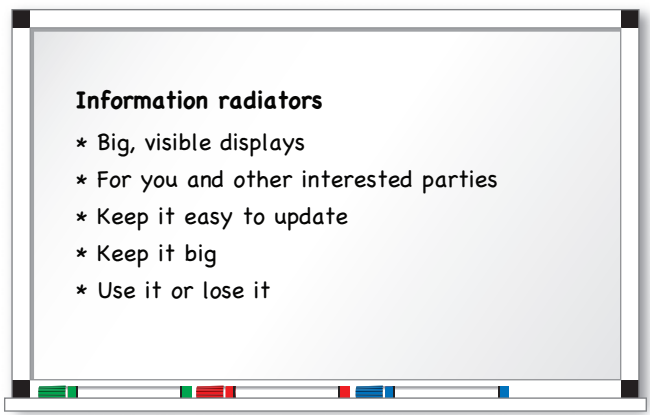
USE IT OR LOSE IT!

Don't use information that isn't valuable to the team or the stakeholders. There are few things more demoralizing than having to update information that you don't see the point of and that no one seems to care about. Constantly reevaluate whether you should keep the radiator as is or if you should change it in any way.

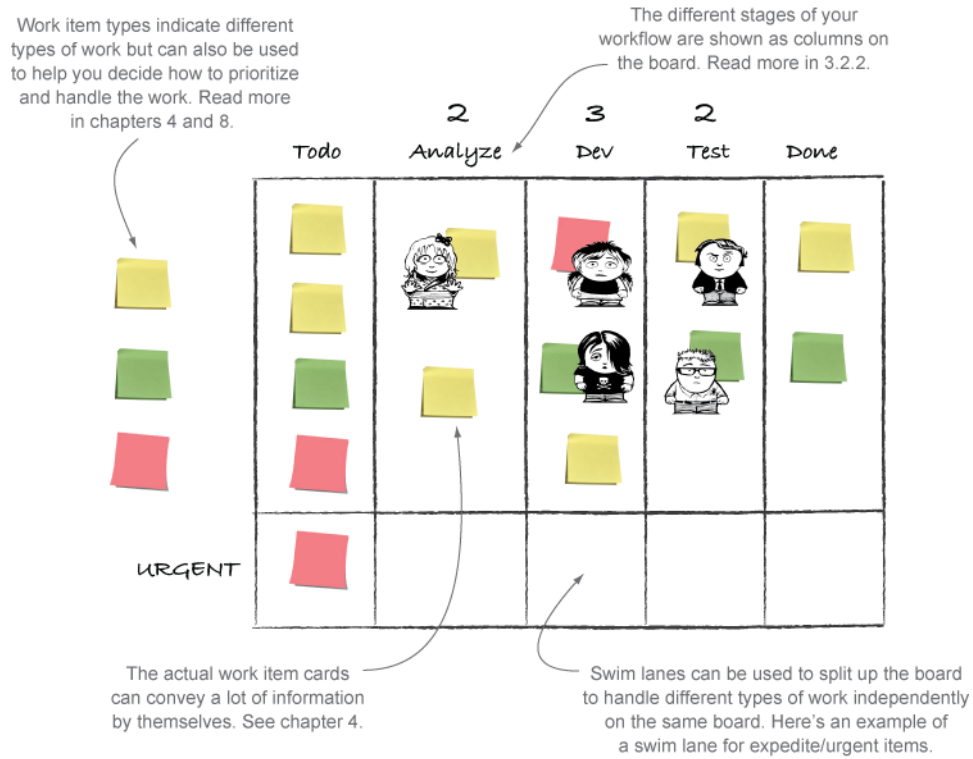
1+1 EQUALS 3

When you combine the power of making policies explicit with visualization using information radiators, the policy is in your face, and it helps you do the right thing. If this isn't helpful enough, your fellow team members will probably apply some positive peer pressure when they can see that what you're doing, or the visualization of what you're doing on the information radiator, is diverging from the policy you agreed on (see section 3.1).

The ultimate kanban combination of making policies explicit and visualization of the work through information radiators is the kanban board. Let's take a closer look at that in the next section.



3.2 The kanban board



When you're working in a team, there are lots of things you need to know in order to collaborate effectively: things such as who is working on what, whether you're focusing your efforts on the highest-priority work, that blocked items aren't left unresolved, and that work isn't queuing up in front of a bottleneck. You want to be able to track the progress of the work. Not only is this important to the team, it's also important to other stakeholders. Why not create a good information radiator for all of this?

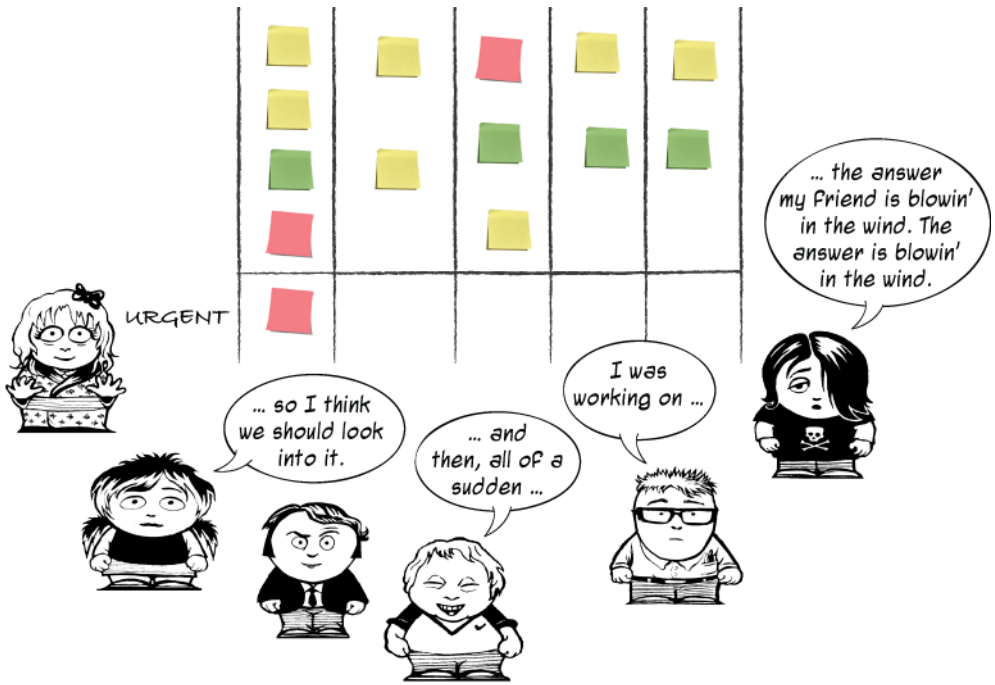
Kanban teams use a lot of information radiators, but one that stands out as the most common and prominent is the board itself.

3.2.1 The board

The basic board is a whiteboard or some empty wall space where you put up stickies or index cards to visualize your workflow and how the work is progressing. Update it frequently, so everyone can see and act on the most current information.

Using a big board not only makes it a good information radiator, but also makes it easy for a group of people to simultaneously move things around, to discuss what is happening, to create new work items, and so on. It's here where you gather around

and "tell the stories" of what happened and might come in the future, like a campfire that the tribe gathers around. The board becomes the natural place for gathering and sharing.



To maximize the reach of the information on the board, you should aim to set it up in a location that is easily accessible for everyone and in which you have enough space to gather for daily standups (see chapter 7) to discuss the work.

Sometimes it's difficult to use a big physical board. If the furniture police or a lack of space is the problem, you can always use a foldable board, a board made out of cardboard that you can at least use during standups. If this is too much of a hassle, if the team is distributed in different physical locations, or if something else stops you, there are plenty of good digital boards to go around (see appendix B).

WARNING You should think twice before going completely overboard with electronic tools. Given the benefits of a physical board, you shouldn't use one or two off-site team members as an excuse to strip these benefits from the rest of the team; using both physical and digital versions and doing a little double bookkeeping is usually a lot less work than you imagine, and it's often well worth the investment.

Meet often: daily standups

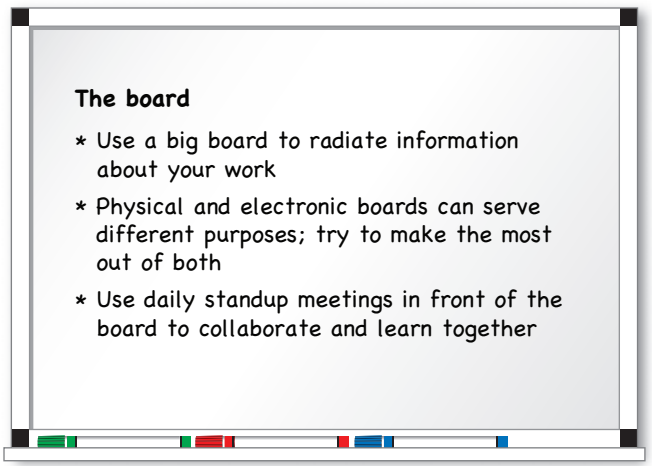
The board is a great tool that radiates your current status to the team. Another great practice to combine with the board is to gather around it for short but frequent meetings. In these meetings, you'll be able to coordinate, share, and learn from each other. Keep the meetings short and to the point; focus on what has happened on the board. This type of meeting is commonly referred to as a *daily standup*.

A simple recipe for a daily standup can be as follows:

- Decide on a recurring time. The earliest available time in the day that isn't painful for anyone is a good starting point so that you can use the meeting to begin the day together.
- Keep the meeting short: a max of 15 minutes. A smart way to make sure the meeting doesn't run long is to stand up during the meeting, ergo daily *standup*.
- Discussions that don't involve most people are paused and moved to after the meeting.

For many teams we have coached, these two practices make a big difference: visualize your work so that you easily can see what's going on, and meet every day to follow up on the work and share problems or concerns. You can learn more about effective daily standups in chapter 7, section 7.3.

OK, so you've got a board or a piece of wall or other means to create a board. What should you do with it? What makes up a board? How do you create it? A lot of questions are piling up. In the next section, you'll find the answers.



3.2.2 Mapping your workflow to the board

If you were clueless about what a kanban board looked like when you picked up this book, you should at least have a hunch by now that it often has a number of columns.[3] The columns represent the different steps the work flows through—but what are these steps? How do you know what to put on your new board?

For starters, you want to capture the *actual* workflow and not the formal, company-standard, on-paper version that you're *supposed to* use. An easy way to get started is to identify one or two of the most typical work items and "walk the stream." You need to find out how the work flows in your particular context.

Getting your workflow correct can be hard; as you recall from chapter 1, the Kanbaneros spent quite a lot of time discussing it. It was time well spent, because it forced them to examine implicit assumptions and gave them a deeper understanding of how they work as a team. Up to the point where they mapped their workflow, there could be many views on how the work was flowing, but making it visual also made it explicit and to the point.

A WORD FROM THE COACH It isn't the manager, the team lead, or some kind of coach who should design the kanban board; the team members who are going to use it should do that. This creates a sense of ownership and increases the likelihood that they will honor the policies and the processes they agree on.

Next, resist the urge to improve your process here and now. The whole idea of visualization is to understand the work and make improvement opportunities more obvious to you. Let the board help you with that. For now, map your workflow. Who knows, the opportunities might not be where you think they are. They might be somewhere else.

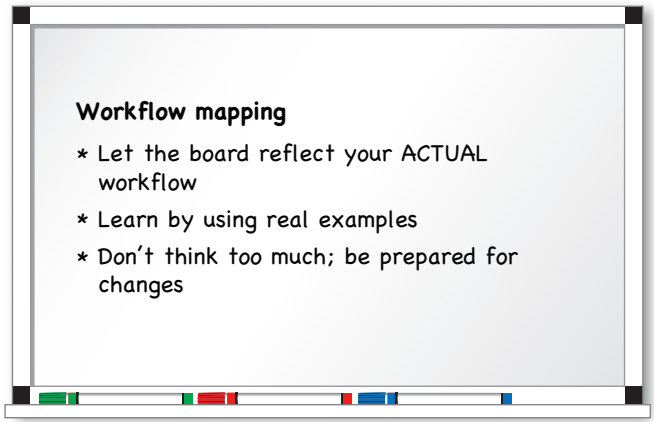
Not all types of work go through the exact same workflow, but that's OK. You have to decide together how to handle this. Perhaps a certain type of work skips a column or two; for example, maybe bugs don't have to go through the Analyze column on the board, as the Kanbaneros decided for their bugs in chapter 1.

Sometimes a column's name might be too specific to suit all types of work, but when you consider what it's about, a more abstract name makes the workflow work for all cases. An example is Test, which makes sense for code but maybe not for documentation or investigative types of work. But when you change the column name to Verify or Reviewed by Peer, it turns out that they both have the same workflow.

Finally, don't put too much effort into this up front; rather, be ready to redo the board when you see that your work doesn't flow as you first thought. We recommend that you draw your board with an erasable whiteboard marker. In this way, you can easily change the board as you see fit. Hold off with the electronic tool and the fancy tape until you feel the design is somewhat stable.

---

[3] To be frank, the board can look however you want it to. We've seen examples of spirals, stairs, and other creative solutions. But the column-based board is by far the most common and probably a good place to start.

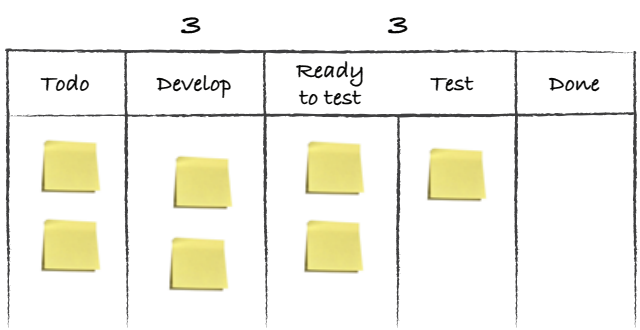


3.3 Queues

Queues can help you to manage handoffs, get a more even flow of work, and give the team members visual signals that work is ready to be started. On a board, this often manifests itself as a separate queue column before or after a column. Here are a couple of examples of queues on a board:

- *Todo*—The first column of the board
- *Ready for Development*—Things that have been analyzed and now are ready to be picked up by developers
- *Development Done*—Items that have been developed and are now ready for testing
- *To Test*—Stuff that is ready to be tested

As you can see from the last two examples, the same state can be expressed in different ways. Development Done and To Test are more or less the same thing. Both of them signal that development is finished, and testers can pick up the items and start testing them. Where you place the queues (belonging to Development or Test, in our example) is entirely up to you. Here's a simple board with an example, to clarify this a bit.



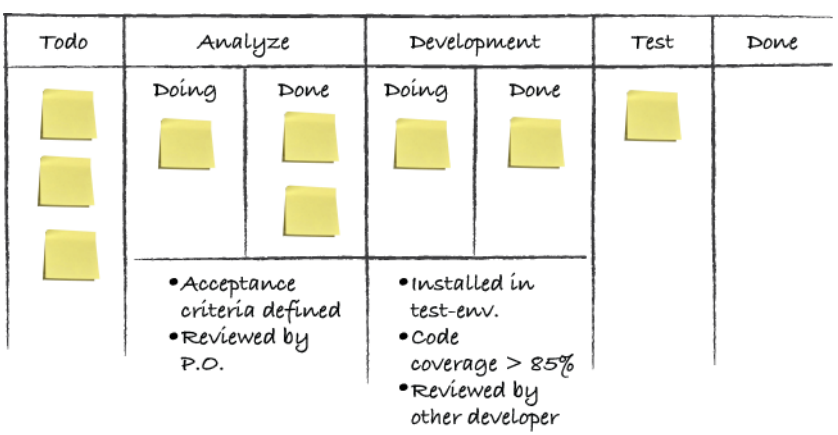
A Done (Development Done, for example) column is the normal case and signals to others in the workflow that this step is finished and someone else can start working on the work item. The person moving it to Done doesn't need to be aware of what's going to happen to it next. For Development Done you could imagine that a tester and a product owner do some kind of quick "triage" together to decide where a work item will go next: maybe into Test, maybe to Stakeholder Verification, maybe to Ready for Production.

A Ready column (like the Ready to Test column on the board we just showed you, for example) is always for a known step, because it's placed in front of a step, ready to be processed by it. Steps like this are used for distinct handoffs, maybe even to other teams. Work is pushed into that step: for example, an Inbox column where work is moved by an event or cadence[4] such as the biweekly planning meeting or when you have three items left in the Inbox column. In our example board, you can see that this team has decided to split the Test column into Ready to Test, and Test. Ready to Test is a queue with work that is ready to be tested. Because the testers allow for three items, you count the items in Ready to Test against that limit.

From this you can see that the testers have two items in the queue (items waiting to be worked on) and another item in the Test column (they're working on that item).

ENTRY AND EXIT CRITERIA

A great way of rooting out misunderstandings and achieving consensus around what the columns, the queues, and the workflow in general mean is to answer the question "Which criteria need to be met in order for me to move a work item to the next column?" These entry and exit criteria can, once they're made explicit, be captured as bullet points or as a checklist and visualized above or below the column they apply to, for example by writing them on a sticky or directly on the board.



[4] The rhythm or heartbeat of your process; read more in chapter 9.

These criteria should then be checked every time you move an item on the board. When you have new or modified criteria, it can be good to use them as a checklist against every item discussed during the daily standup, maybe even adding the policy that at least two people have to agree that the criteria are met before you're allowed to move the item. The policies are continuously discussed and incrementally changed and improved. Just as with the columns, you could have entry or exit criteria, or both, depending on how and what you want to check as each work item moves through your workflow.

A WORD FROM THE COACH We often find it useful to revisit the entry and exit criteria when something related to the workflow comes up: for example, in conversations during the standup, in team retrospectives, or when doing root-cause analysis for a defect. You ask, "Did we follow the criteria?" If you did, is there anything you could add to the criteria that would help you avoid this issue in the future? If you didn't, you should examine why you're not following the policies you've agreed on together.

3.4 Summary

This chapter talked about visualizing your work—making work and the policies around it visible where previously they were not:

- Visualizing your work is ultimately about creating the transparency and information-sharing needed to understand how the work works and to collaborate effectively together.
- The process of visualizing work means making information visible that previously wasn't, making implicit knowledge and policies explicit, and, in doing so, resolving inconsistencies and conflicts that surface.
- The kanban board is a great way to visualize your workflow and share information about priorities, who's working on what, the progress of individual work items, and so on.
- By using a big, visible board, the information radiates to everyone interested instead of being hidden in people's brains or inaccessible tools.
- Creating a kanban board is easy: you map the workflow of your work items—that is, the steps they typically go through in order to be completed—to columns you create on a whiteboard or a wall.

The next step is to create the work items, which is the subject of the next chapter.

4 Work items

This chapter covers

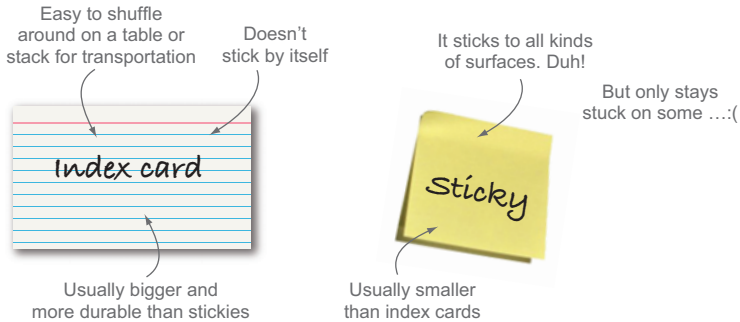
- Work-item cards
- Design principles for work-item cards
- What to keep on work-item cards and how to use this information to gain better knowledge about how your work works

Work in the software business is often not visible. Most of the work takes place in our heads or inside the computer. In order to get a better overview of who's working on what and the status of the work, you visualize the work—thereby making information visible that previously wasn't.

By far the most common way to track work items is to create a small card that represents the work that is being done. It can be an index card or a sticky note[1]: anything that is easy to work with and move around on a board, such as a white board. Cards on a board are a simple yet powerful way to see progress, bottlenecks, and queues happening in your workflow—and to make it in-your-face apparent to everyone what is happening.

[1] For example, a Post-it® Note.

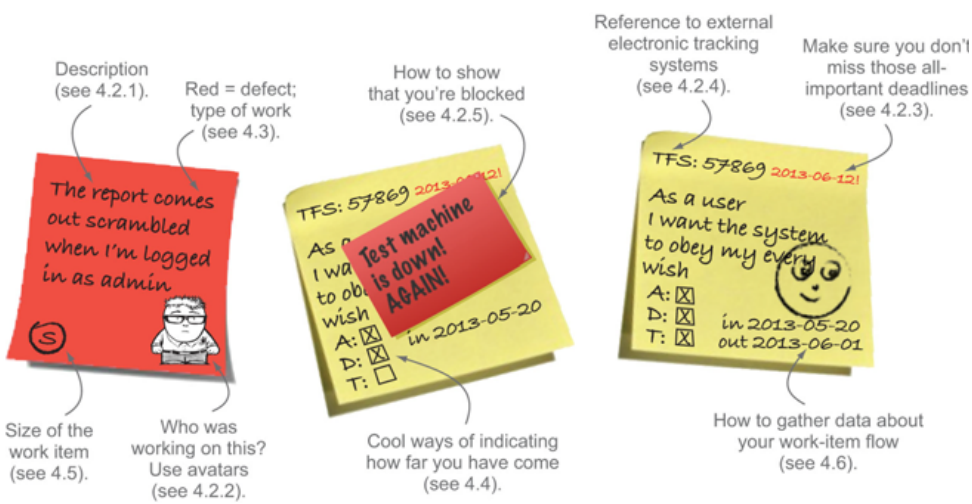
Using a physical card gives you some advantages over an electronic representation. A physical card can easily be annotated and customized with avatars, blockers, progress bars, tracking IDs, and other things that we'll look into in this chapter. Also, because cards are tactile, they're easy to use for collaboration, to move around, and even to take with you if needed.



This tactile concept holds some other secrets. By moving stickies on a board, you involve more of your senses in the process and thereby create a stronger connection between you and the work items. It's more likely you'll remember that you took responsibility for an item that you moved with your hands from one column to another.

This chapter is dedicated solely to those cards. What goes on them? How do you show that a work item is blocked? Who's working on this item? These questions and more will be answered in this chapter. As always, don't limit yourself to only our suggestions; you should probably not use all of them at once, and there are probably other things that could go on a card that we haven't thought of.

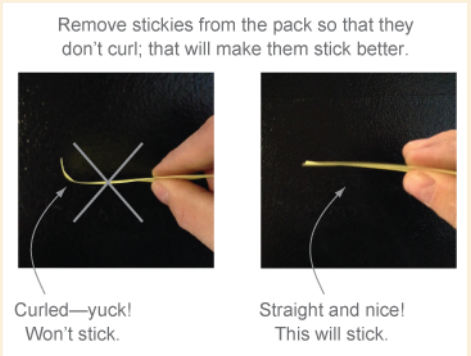
Use your imagination to solve workflow issues, and use our suggestions as inspirations to improve on. The topics we discuss in this chapter are common ways of visualizing the basic information of a work item. Make sure your card contains all the information needed to help your team make decisions regarding each work item.



How to remove a sticky note from the pack

Yes, this might be the geekiest sidebar in history.

But it's here to protect you from a plague that haunts a lot of teams using stickies as work-item trackers. Picture the scene: you enter the office and see a board with a lot of stickies lying on the floor beneath it. You happily realize that it's not your board; your stickies are securely attached, thanks to this sidebar. With a wry smile on your face, you visit the poor team whose board is missing its stickies to educate them in the noble art of "how to remove a sticky correctly."



The trick is to be careful that the sticky part doesn't curl. Slide your finger under the note up to the sticky area, and then lift it from right to left, slowly but firmly. That will make the sticky part straight and nice, and thereby increase the sticky area of the note.

Now you know and can pester all your friends with the proper way to remove a sticky. Not only is this a geeky party trick, but it will also, more seriously, save you problems with work items falling off your board.

Instead of a physical work-item card, you can use electronic systems that represent the work. There's nothing wrong with that, and many of the systems nowadays are great, especially for gathering data and reporting metrics. But remember that you're giving up some of the benefits of a physical, tactile card. When in doubt, start physical and then move to an electronic version later if you see the need. Or even better—use both!

4.1 Design principles for creating your cards

The work-item card can be represented in a lot of different ways, and we'll soon dive into some of the common patterns for doing so. You will undoubtedly find other ways that suit your needs better, so please change and elaborate as needed. But there are a couple of design principles that you should keep in mind when you're deciding what to put on your cards and what to annotate them with. Let's take a look at some design goals for creating work-item cards.

4.1.1 Facilitate decision making

First and foremost, you want the design and information on the card to facilitate decision making in the team. Strive to help yourselves self-organize. You want to avoid situations in which the team doesn't know what to do next and they have to ask someone in order to continue. Equally bad is a team in which people respond differently to the same situation because of lack of information. This is one of the main reasons for having explicit policies and clear ways of working in the first place.

These goals are easy to state, but it might take some time before the team knows them by heart. Meanwhile, strive for simple, obvious rules, and make them visually apparent to everyone on the card. An example could be to show with an avatar who is working on an item so that you know who to talk to for more information or which items don't yet have someone working on them.

4.1.2 Help team members optimize outcomes

The design of the card should also help team members optimize the outcome when it comes to matters such as risk, customer satisfaction, and economics. If a work item is high risk, you need some way to show that to the team members, so that risky actions can easily be avoided. An example of a high-risk work item is one with an explicit deadline for when a new law goes into effect. Missing this sort of deadline could be associated with a fine of some sort. Therefore, you want to make sure you all know about the deadline and prioritize accordingly.

The same goes for customer satisfaction or economics; make it apparent if one (type of) customer is more important than another. An example of this is a work item with new functionality that greatly benefits new users. Maybe you want to complete that before any maintenance work items.

How to treat the work item is governed by policies (explicitly articulated or not) and often associated with the type (for example, normal features, defects, and maintenance work items) and class (for example, work items that are particularly urgent or have a fixed delivery date) of work. We'll be talking more about work-item types in section 4.3 and in chapter 8. Help the team to follow the policies by clearly showing what type and class of work they're picking up.

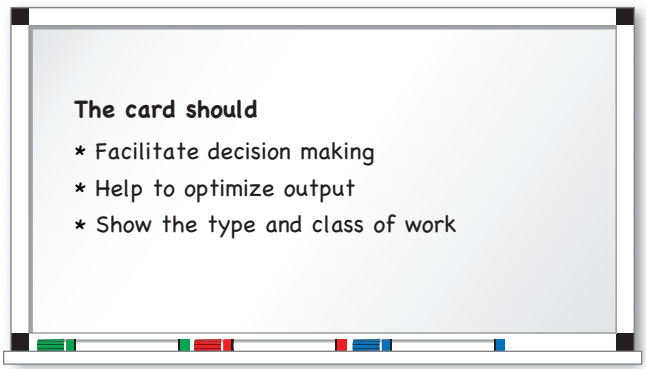
As you can see, there's quite a lot to think about, but let the guiding star be *simplicity*. There's no use adding too much on the sticky, if you stand the risk of forgetting what it was about. Start simple, and experiment with more information as you see the need. You want the information that the work item is radiating to be easy to see and understand. Along those lines you also want changes and *smells* (see the sidebar "Process smells") to be apparent and clear. You want to act on things that are outside of normal operation, and it must be easy to spot them as they happen.

Process smells

We borrow the *smell* concept from Kent Beck and Martin Fowler, who talk about code smells (*Refactoring: Improving the Design of Existing Code,* Addison-Wesley Professional, 1999, http://amzn.com/0201485672). A smell could mean trouble, but it isn't necessarily trouble because it smells. Much like the smell from a toddler's diaper, if it smells, you should at least check it, but it doesn't mean you'll always find something there.

If one of your stickies has a smell to it—for example, that it has been sitting idle in a column for a couple of days—you should check on it. It might be that you need to take action. But there might be perfectly natural reasons for it to be stuck there.

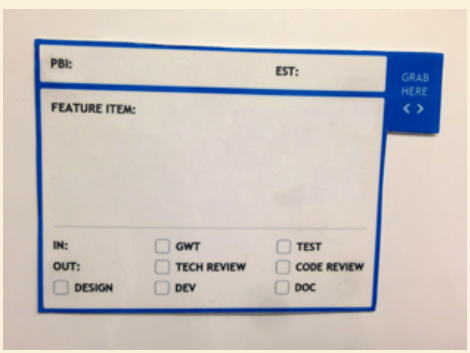
A WORD FROM THE COACH Keep these design goals in mind as you read the patterns on work-item design, and see if you can come up with other ways of articulating the goals in your team and on your board.



What about the size of the card?

As you soon will see, there can be quite a lot of information on a work-item card. Be sure to choose a card size that allows you to fit all the information you need on the card. Stickies come in different sizes, so experiment with sizes that suit you.

We've also seen a lot of teams that create cards of their own with custom-made structure and form. Here you can see a refined work-item card;[a] the team has laminated the card in plastic so that it can be wiped clean and reused over and over:



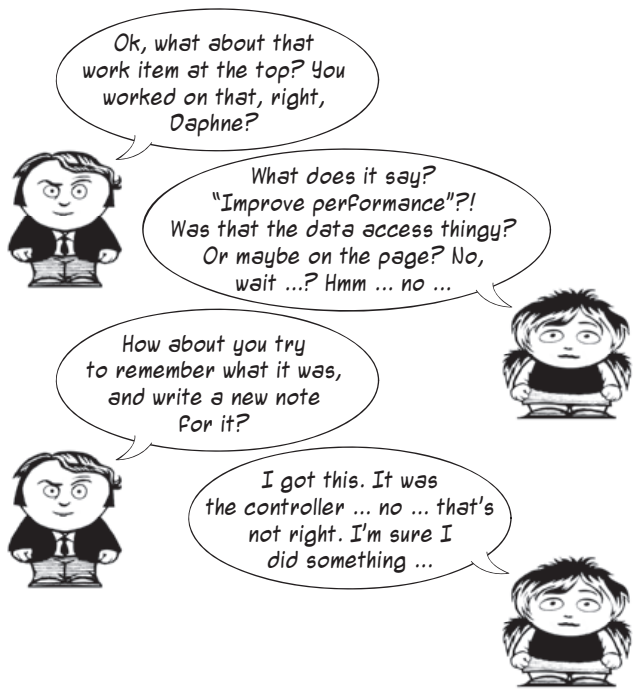
There's no right or wrong. Just pick a size and form that suits your needs, and change it as your needs change.

a. One of Marcus's clients, Tradera (Swedish eBay branch), had these cards on a board when Marcus visited. They're some of the most elaborate cards we've seen.

4.2 Work-item cards

A lot of things can be on a card, and in the following sections we'll examine some common attributes, starting with the most obvious one: the description of the work item.

4.2.1 Work-item description



To many of you, this might sound like a no-brainer; *of course* the work item has a description. But this is one area in which we see that many teams have room for improvement. We can't count the number of times we have seen teams argue over what an item is about because the description on the card is inadequate.

In order to more easily talk about the item and its content, the description needs to be terse, to the point, and easy for everyone on the team to understand. What does that mean in practice?

USER STORY

A user story is an example of a description that is short and terse and can prove useful.[2] It says *what*, for *whom*, and *why* the work item is being worked on.

[2] Beware of relying too much on the user-story template format and its merits. It's just a way to structure your description of the work at hand and not a silver bullet that always works. Some (including Joakim) have been known to call them an antipattern from time to time.

We won't dwell on the user-story subject here, because it's not in the scope of this book.[3] For now, we can simply say that it's a Card with a little text that is a reminder of a Conversation you're going to have later. In the conversation, you'll flesh out the details and write down your Confirmation as acceptance criteria. That can easily be remembered with the acronym CCC: Card, Conversation, and Confirmation.

A common template for a user story is

As a [role], I want [feature], so that [benefit]

Or maybe

In order to [benefit], as a [role] I want [feature]

Use a format that feels natural to you, but make sure to include the Why (benefit), the Who (role), and the What (feature) in the description.

TITLE

If the description tends to be longer than you care to repeat every time you talk about the work-item card, you could benefit from adding a title to the card. A short sentence that is easy to remember and to refer back to can help you remember what the work item is all about.

"THE ONE WHERE ..."

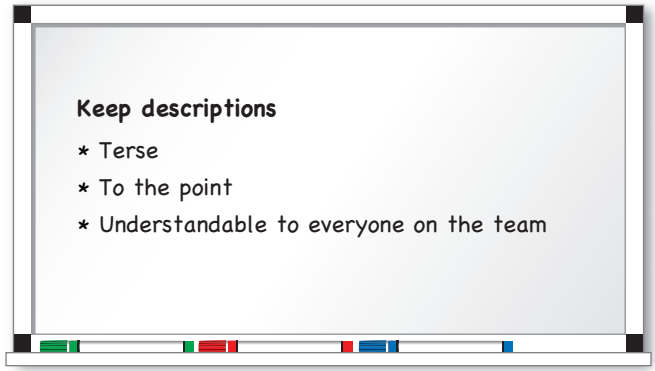
User stories are one way to write descriptions, but sometimes they don't sit right with the team for the item at hand (like bugs, for example). You still need to have a good description, and a simple mnemonic that we've used is the *Friends* naming convention: *in your mind,* put "The one where" before the description on the card. (All the titles of the episodes of *Friends* started this way, such as "The one where Ross and Rachel think they're in/out of love again.")

Using this little reminder will put the focus back on why this work item is up on the board in the first place. Make the title easy to refer to in a conversation, like these work items, for example:

- (The one where) the name field allowed too many characters
- (The one where) you were missing the administration rights, if you logged in as administrator

A WORD FROM THE COACH Remember that "The one where" doesn't need to go on the card. It's a little lead-in, there to get you focusing on the real reason behind the work item.

[3] See more in the excellent book *User Stories Applied* by Mike Cohn (Addison-Wesley Professional, 2004, http://amzn.com/0321205685).



Here are a few good (and some bad) examples of what to put on your card as a description:

After learning what the work item is all about, the next problem to address is probably who is working on it now. Let's check out a common solution to this problem.

4.2.2 Avatars

You want information about who is responsible for a work item to be as clear as day, so you know where to go with your questions, suggestions, and praise. Many teams indicate who is working on what by attaching an *avatar* to the work-item card. In this context (see the sidebar "What's an avatar, anyway?" for the real meaning of the word), it's a clear indication of who is working on what.

The reason many people use pictures, cartoons, or drawings of themselves is because it's easier to identify a person using *pattern matching*. It creates an instant connection if you compare an image to a person, as opposed to looking at a scribbled name or signature that might need some translation to read and then to associate with the right face.

This means you should use avatars that resemble your team members or are caricatures with features that are easily recognized. Of course, you could add a key or legend on the side of the board, but that means people would have to look up which avatar belongs to whom. With avatars that resemble the team members, you don't have to do that lookup.

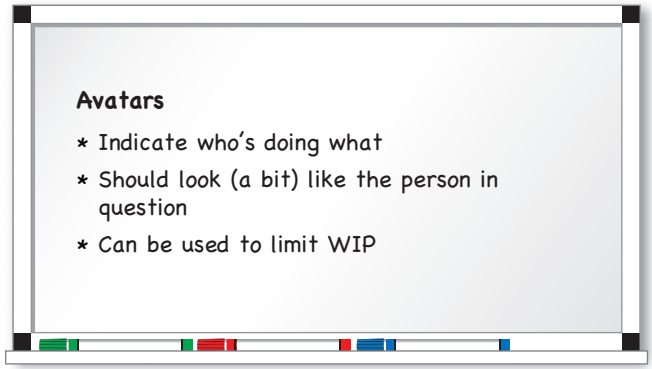
What's an avatar, anyway?

In computing, an *avatar* is the graphical representation of the user or the user's alter ego or character. It may take either a three-dimensional form, as in games or virtual worlds, or a two-dimensional form, as in an icon in internet forums and other online communities. It can also refer to a text construct found on early systems such as MUDs. It's an object representing the user. The term *avatar* can also refer to the personality connected to the screen name, or *handle*, of an internet user, such as the user icons on Facebook, Twitter, and other social media sites.

Some teams use avatars to limit the work in process for each person. This can be an effective self-constraint for hoarders[4] who have a habit of being involved in every item on the board. For example, giving each person three avatars to put in play on the board is a visual and effective way to make sure no one takes on too much work.

A WORD FROM THE COACH You might think having cute animals or cool cartoons as avatars is the thing and will spice up your board. Our experience tells us you're probably wrong. One team that Joakim coached decided to use dog avatars. This caused confusion when they had to try to remember who the poodle was, why the Dalmatian had not completed those tests yet, and what on Earth the schnauzer was doing with that analyzing task all that time. Use avatars that at least resemble the person in question!

[4] Nickname for people who often end up collecting a lot of stickies.



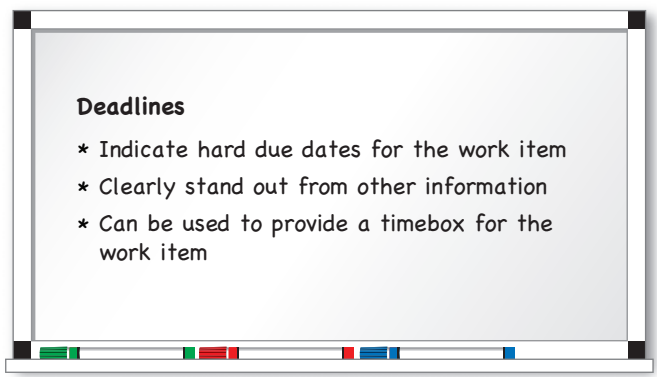
By using avatars, you can easily see who's working on the item in question. We now turn our attention to deadlines, which help you know when the work item is due to be completed.

4.2.3 Deadlines

Deadlines can exist for a number of reasons; there can be features that need to be done for a campaign, a new law kicking in on a certain date, or a new customer coming to the office next week, for example.

To clearly show the date when the work needs to be done, you write it directly on the work-item card, maybe even with another color that stands out. Deadlines are *risk-management information* and help the team prioritize and self-organize; you don't want to miss seeing them, so make sure they're clearly visible.

Some teams even use different colors of stickies for fixed-date-delivery types of work items in combination with the deadline date. Make sure you use the same visualization every time to create a pattern or habit. Always use the upper-right corner of the sticky for deadlines, for example.



There—you now have deadlines on the work item too. They're another piece of important information. Continuing down this road, putting all information on the card, you'll soon run out of space. What do you do about information that you cannot fit on the card? That's the subject of the next section.

4.2.4 Tracking IDs

We have talked a lot about the good things that a physical board with work-item cards on it brings, but there are some things that such a setup doesn't do well. The limited space[5] on a sticky might not hold all the information needed in order to complete a feature. There might be a lot of other documentation that simply isn't feasible to try to attach to the sticky or keep near the board. Or you might be required to track your work, the number of hours, and the progress in an electronic system.

In these cases—and there are surely other cases too—you need to easily know which item in the electronic system represents the one on the board. You can think of it as a "more information here" link.

A simple solution to this is to write down the electronic tracking ID in a corner of the sticky. Similar to the deadline, make sure to use the same corner on every sticky, which creates a good habit. Or you can prefix the tracking ID with, for example, *JIRA:*, *TFS:*, or *Git:* to make sure it's not confused with other numbers that might be on the note.

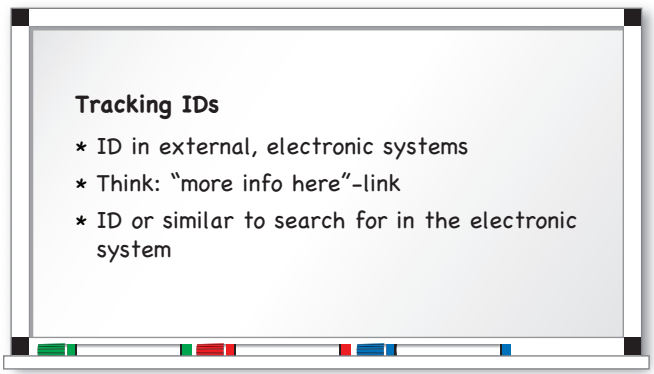
WARNING Beware of falling into the trap of putting all the information about the sticky into the electronic tracking system, regressing back to work-item cards with no easily understandable description (see section 4.2.1).

Astute readers will now begin to worry about the "double bookkeeping" this will make you do and how to keep the board and the electronic system synchronized. To handle that, you could first decide on a master—preferably the physical board, because it's "out there" and the one on which your work is most visible.

[5] Limited space can be a good thing, forcing you to work with smaller items and divide larger items into many smaller ones. This is the reasoning behind writing user stories on a small card, for example. If you feel that a big index card is too small to hold your information, try an even smaller one to force yourself to express it briefly.

In the electronic system, you can then make a simplified version of the workflow: Not Done, and Done, for example. This will make the updating much more lightweight and not create a lot of extra work. Also, on the work item in the electronic system you can now attach documents and links to other important resources.

Of course, you can go the other way around and use the electronic system as the master; but then we recommend that you try to get a projector. Or why not use a large touch-screen display and a board-like plug-in for the system in question?[6]



Until now, we have considered the evenly paved road of success. That's not the road we all travel in reality. What happens when things don't work out as planned? Let's investigate that next.

4.2.5 Blockers

Even though the goal is that your work flows quickly and smoothly through your workflow, sometimes things happen: you have to wait for someone else to complete their work, someone is sick, or you run into questions that hinder the work from being completed.

BLOCKED ITEMS

In these cases, you want the item to stand out and show that it's *blocked* from further progress. When you do so, you get a signal and a reason to focus on the blocker (at your daily standup, for example). Eyes are drawn to the items that are different from the normal items, and this helps you to focus on those deviations. A blocked item should be a visual *smell*, to use the terminology we introduced earlier (see section 4.1.2).

To accomplish this, many teams attach another sticky on top of the blocked work item. This is a good idea because you also can write the reason why the item is blocked

[6] There are loads of these plug-ins, but a few that we like are Kanban for TFS, JIRA Agile, and HuBoard for GitHub.

on the blocker sticky. In this way, you get not only a signal that the work is blocked, but also some information about *why* it's blocked, which in turn helps the team focus on resolving the blockage.

There are many other common alternatives to signal that an item is blocked: magnets put on top of the blocked work item, turning the work item on its side, or moving it into a blocked "parking lot" on a separate part of the board. If you decide to move the card, make sure you have some way of remembering its previous location, so you can move it back there once the blockage has been resolved.

A WORD FROM THE COACH Although keeping a separate "parking lot" for blocked items might seem like a good idea, we advise against it. It's basically the same thing as saying that it's OK to be blocked—"Look, we even have a dedicated area on the board for it!" Keeping the blocked item in its column keeps it in your face, affects your amount of WIP, and forces you to constantly have to consider it during standups.

BLOCKAGE PROGRESS

Some teams also write a simple form of progress indicator (see section 4.4) on the blocker sticky, to put further focus on resolving it rather than to leave the item blocked. For example, a blocked work item with three dots on it means the sticky has been blocked for three days, and that needs your attention, right? Some teams use more elaborate policies for this: for example, after three days they contact the person or team blocking them in person, and after five days they escalate to management or another support function.

When choosing which blocker design works the best for your team, remember that you want the item to stand out from normal work items and preferably radiate some information about why and for how long the item has been blocked.

Taking smells to the next level

During the Lean Kanban North America conference in Chicago 2013, we learned about a team that has taken visualizations and smells to the next level. Instead of adding a small sticky to the work item to indicate that it was blocked, this team staples a real banana peel to the index card.[a] As the days go by and the blocker isn't removed, the banana peel gets darker and darker, and an unpleasant smell starts to spread. That's a call to action for you!

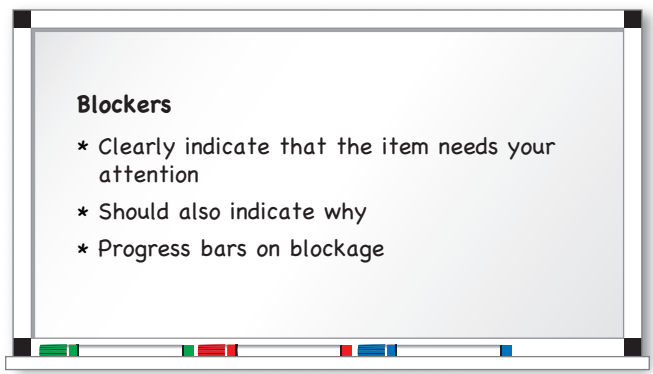
a. Image courtesy of Gualberto107/FreeDigitalPhotos.net.

(continued)

Their next idea was to use actual bugs to indicate software bugs. We don't know about you, but we would certainly beef up our code quality to avoid having to nail real bugs to the board.

Some teams love to get creative with these things. If you're in a team like that, you should by all means go bananas with the visualizations …

With blockers, you can now handle situations in which the work doesn't flow as planned.



Up to now we have used the generic term *work item*, but work items are of different kinds or types. Let's see how you can visualize that.

4.3 Types of work

Work items can be of different types, such as bugs, technical items, or maintenance or feature requests. Work of different types should often be handled and prioritized in different ways. You want the team to be able to easily distinguish different types of work from each other and be able to prioritize work items on their own.

To help the team self-organize around how work should be handled and prioritized, a common practice is to let each type of work have its own color. In this way it becomes easy to distinguish a defect from a normal feature, for example. These types can also be used to set up policies on how work should be treated, commonly referred to as *classes of service* (see chapter 8).

Here are a few common examples of types of work. Some of the colors used here have grown into a convention in the kanban community. But, as always, make sure the colors make sense to you and your team. (Although the stickies are shades of gray in the printed book, they're green, yellow, and red in the eBook and on the kanban board.)

The choice is up to you, but you should take care to make sure that not everything on the board looks the same. You don't want to end up in the yellow sea of stickies, which makes it hard to know how to prioritize between different types of work items or to see any kind of patterns.

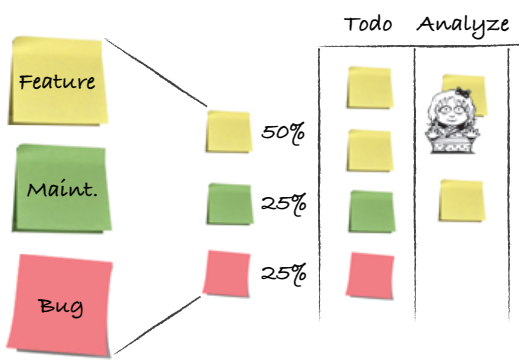
Making work of different types apparent—by using different-colored stickies, for example—can help you see what's going on. Simply by glancing over the board, you can get a feeling for the overall status of the team. For example:

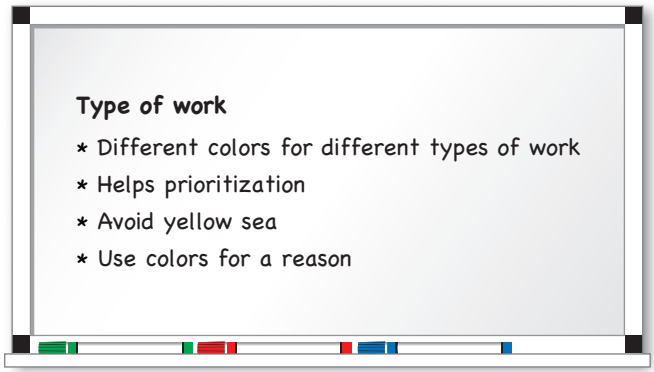
- If you see a lot of red stickies (bugs and defects), you can see a problem with quality in your system. You should probably make an effort to improve quality.
- When you note that there are no green (maintenance, technical) items on your board, you might not be paying off your technical debt as needed, which makes it harder to maintain the system over time.
- No yellow items (features) means you're not adding new features to your system. This might be what you want, or not. The board gives you a simple visual signal that tells you how it is, either way.

If you only used yellow items, you would have to read each and every one of the notes to know the overall status of all the items on your board.

WARNING Don't fall into lazy mode and grab any colored pad of stickies you have lying around. Mixing colors of stickies on the board for no reason causes confusion. The best thing is to pick a limited number of common types of work, assign each one a color, and then use that color all the time for that type.

If you have a hard time remembering what type of work each color means, a legend can be posted on the board to rule out any misunderstandings. This can also be helpful for stakeholders and others who don't work with the board on a daily basis.





Let's now turn our attention to seeing how far you have come. In the next section, you'll get a simple tool that will help you track that.

4.4 Progress indicators

A card in a column on the board is a great visual cue and gives you a lot of information, but the history of the work item isn't as easily tracked. For how long has this card been sitting here? Is this a normal lead time for the development stage? Questions like these aren't easily answered by looking at the card where it's sitting today.

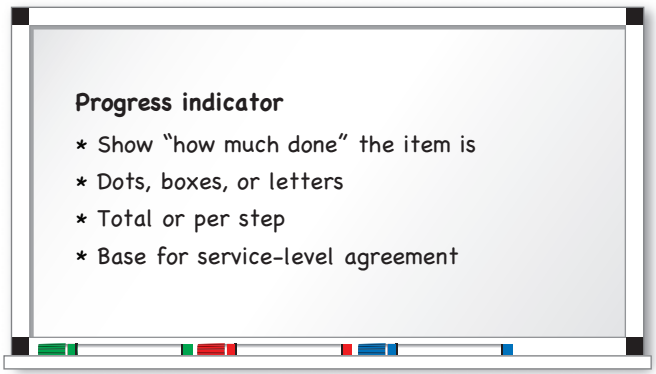
A *progress indicator* is a simple tool that helps you track this information and shows "how much done" the item is. It can be as simple as marking each sticky in the workflow with a dot for each day it's been worked on. If you're a bit more advanced, you might use different colors of dots for different states in your workflow.

In teams we've coached, we have also seen the expected timeline drawn as boxes, which are filled in as the work progresses. This gives you a hint as to how you're doing compared to what you usually do or the expected outcome. You could even talk about using a simple form of service-level agreement (SLA) here: "You can expect items that you classify as 'small' will be done within three days."

For items that don't follow a linear flow,[7] each state in the workflow could have its own box to tick when work there is done, and that will show what's left to do.

GOING GOOFY: COUNTING DOWN

Instead of counting the number of days you've spent, you can instead count down by tracking the number of days left before the item needs to be completed, based on its deadline, for example. Then at the daily standup you can update this figure to reflect the number of days that still remain. If you have been doing Scrum, you know this is a common way to track progress against the estimate on a burn-down chart.



By using progress indicators, you've now started to collect and track data around how your process is working. In order to do that with any accuracy, you'll soon need to know how much effort goes into each work item. Let's take a look at one way of doing that.

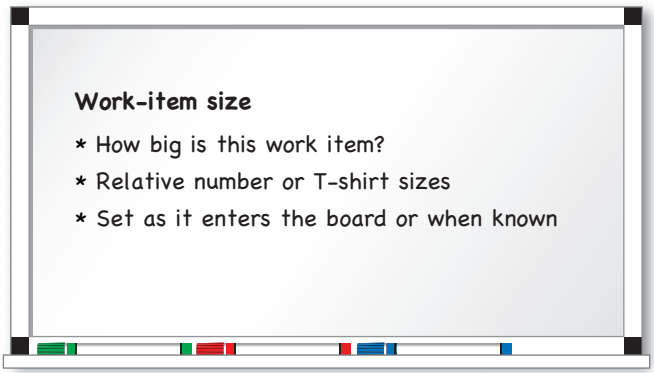
4.5 Work-item size

The size of a work item can provide valuable information to help you manage the work item on the board. The only problem is that the exact size of work in your business is often not known until you're done.

We'll talk more about estimation later (see chapter 9), but briefly, we can say that giving a number for how many hours a certain item will take is hard and often ends up being wrong. A much more compelling approach is to instead compare that work item to other work items. The question is then, does work-item A require more or less work to complete than work-item B?

[7] Where the process doesn't follow a natural sequence of steps, so that the steps to complete the work for the work item can be run in any order.

By doing this for all of your tasks at hand, you can assign equally sized work items a number—what are known as *story points*—or even T-shirt sizes (S, M, or L). These numbers only show your *relative estimates* of how much effort is needed to complete the task. This approach isn't exact, but it's probably more useful anyway, because exact measurements (104 hours, for example) are also guesses.



However you have come up with your estimate, you want to write it clearly on the work-item card, so that it's plain and easy for everyone to see.

4.6 Gathering workflow data

The last few sections have talked about adding information to the work item that can help you collect good data about how your workflow is behaving. This can be useful information for improving your process going forward. Let's take a closer look.

4.6.1 Gathering workflow metrics

When the workflow is set up in a detailed way, you have an excellent opportunity to gather data about how well your work flows and to measure and track trends in your work. In short, you can learn even more about how your work works.

Some of these metrics can be captured quite easily at almost any time, and others must be done as the work progresses through the flow because some of the data cannot be captured in hindsight. Many teams track a couple of easy-to-get metrics, such as the lead time of each work item and throughput (number of items completed per week, for example), in order to learn about their work. In chapter 1, you saw the Kanbaneros setting up such metrics for their process with a few simple steps.

This section isn't about metrics and how to use them. You can read more about metrics in chapter 11. Here we want to suggest a couple of simple ways that you can customize your cards to catch metrics.

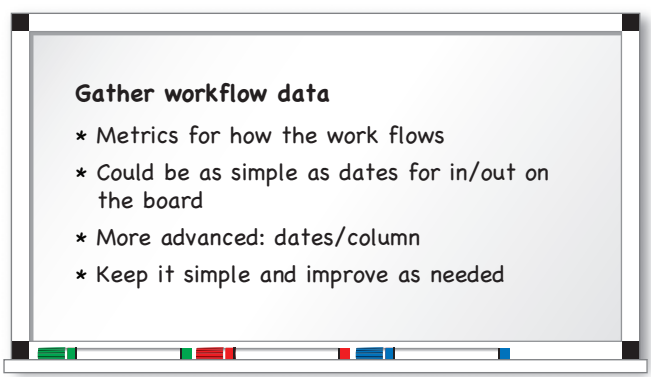
In the simplest form, you can "stamp" the card with the date it enters the board and the date it enters the final stage of the workflow. This will help you track the lead time for that item.

Stamping the arrival date for each new column (that is, each new step in your workflow) can extend this idea. You can now start to see trends in cycle time for each step of the workflow, where your bottlenecks are, and where work often is waiting. You need data like this to build more advanced diagrams (see chapter 11).

Combined with the work-item size we talked about before (see section 4.5), you can now start to make some predictions, such as these:

- "If you put a small work item into the inbox, it will be out the door in three days, tops."
- "A medium work item will be handled in five to eight days, in 9 cases out of 10. We have statistical evidence to back that up."

Remember—you're gathering this data for the *team* to approve. Don't overdo it. Start with a metric that is simple to track, and then make it more advanced as the need arises. You don't want to add a lot of extra work to catch and handle the metrics. Just jot down a simple date on the card—that's about it. Change it when that simple metric doesn't suffice anymore.



This section talked about gathering hard data about how your work items move through your workflow. We'll end this section with a softer and cozier tone: talking about emotions.

4.6.2 Gathering emotions

Some teams use the work items to keep track of how happy the team is with the work they're doing. You can too; simply draw an icon, such as a smiley, an angry face, an indifferent face, or something else indicative of your emotion with regard to the work, on the sticky when you finish it. This little icon indicates how you felt at the time.

Because the card is about to be moved off the board, you don't have to worry about messing up the sticky. Just draw the icon over the existing information as a quick reminder of how you were feeling.

Anyone on the team can capture the "emotion" of the card as it moves off the board. One way to do it is to have a quick vote at the morning meeting. You could also decide that the last team member that touched the work-item decides. In teams in which one or a pair of developers moves the work item through the entire chain, they decide the status at the end.

At regular intervals, you examine the data together to see if it tells you anything about the team and the work. Can you see any trends or patterns? Maybe work of a certain type, or for a certain customer, seems to make the team sadder than other types of work. Could you do anything about that? Examining the data can be triggered by having a certain number of finished items (for example, after every 10 items); or it can be scheduled with a particular cadence, such as every two weeks; or it can be done in retrospective meetings if you have such meetings (see chapter 10).

Fun and maybe useful practices

We have heard about a lot of other practices—too many to list them all in this book. But here are two practices that are fun and a bit different that you may find useful and that might inspire you to try something different:[a]

- Split an item in two parts (literally, with scissors) when you have concurrent activities/parallel lanes that merge together again. For example, a work item goes to security and software review in parallel, and each has its own workflow/column before they come together again
- Use different-sized work items for different sizes of work, so that large is a really big sticky (the roughly A5-sized ones), medium is a rectangular sticky, and small is a square one, for example. You could even have physical WIP limits (by width and height on the board) for these instead of a number.

a. If you do, please tell us about it.

4.7 Creating your own work-item cards

Now you've seen a lot of tips about what *could* be on the card. It's time to get practical and start thinking about what your work items should look like.

We suggest that you do this as a team exercise. Start small and simple, and expand as the need arises. Here are a few things your team can discuss:

- What information is needed for you to know what to do with the work items? Remember the design goals from section 4.1.
- What information may be interesting to other people who aren't working with the work items every day?
- Do you have different types of work? Do you benefit from distinguishing between different kinds of work?
- Do you need blockers? If you do, what are your policies around that? What should happen with items that are marked as blocked? Who is responsible for unblocking them?

As a final little fun exercise, create your own personal avatars. Remember not to go completely overboard with creativity, though; you want avatars that at least resemble yourselves.

4.8 Summary

This chapter was all about the work-item card and the information it radiates (communicates) to you as it sits on the board. A work item should contain all the information the team needs to be able to know how to work with it. We discussed the following:

- *A description*—So you know *what* the work is about
- *An avatar or other marker*—So you know *who* is working on the item
- *Deadlines and other important dates*—So you know *when* you need to have the item done
- *Tracking IDs or other references to an external system*—So you know *where* to find more information
- *Blockers*—So you can pick out items that are blocked and therefore *hindered* from further progress
- *Type of work*—So you know the *type* of work for each item, which is important so you can prioritize the work items against each other if needed
- *Progress indicators*—So you know how much of the work you have *done so far*
- *Size*—So you can see the differences in *size* and effort
- *Flow data, emotions, and other data accumulated during the flow through the board*—So you know how the *work behaved and how the team felt about it*

If you find this list daunting—stop! Don't do all of it; we haven't met a single team that does all these things at the same time. Start with what fits your current needs, and don't overdo it. Add features as needed.

If you find that something is missing from this list, you're probably right, and that's how it should be. Don't limit your work-item cards to this list alone. This is a starting point. Feel free to elaborate with your creativity and come up with amazing things.

Here are examples of work-item cards that use some, but not all, of the features we've talked about:

Now you have the tools you need to create a lot of work items; but you shouldn't. You should create small amounts of work. This is the topic of the next chapter.

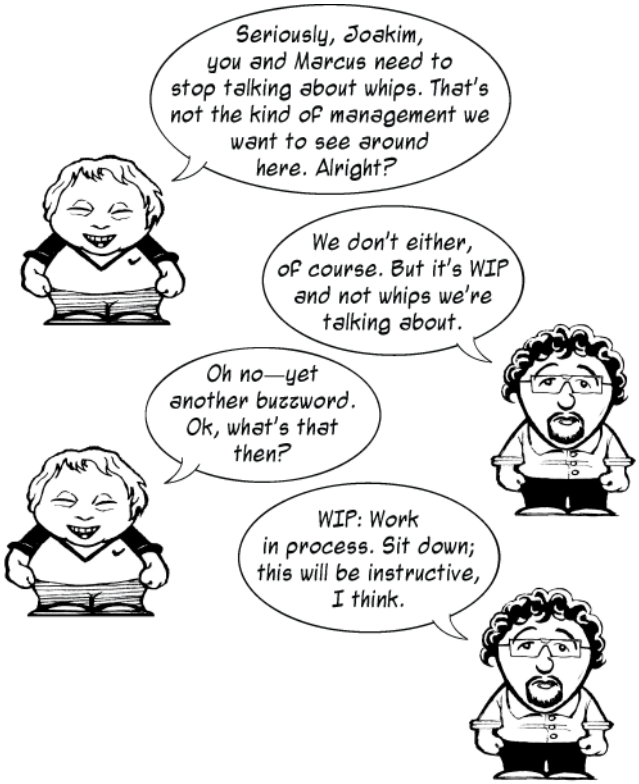
5 Work in process

This chapter covers

- Introducing the concept of work in process (WIP)
- The effects of a lot of work in process
- How to limit work in process

Work in process (WIP) is a phrase that you'll hear a lot in the kanban community or when reading about Lean. WIP seems to be something that you don't want or at least want as little of as possible, so you often hear kanban aficionados talking about "limiting" WIP.

This chapter will help you to understand what WIP is, what could happen if you allow a lot of WIP in your process, and, finally, some ways to help you to limit work in process.



5.1 Understanding work in process

In this section, we'll dissect the concept called work in process. First let's talk a bit about the abbreviation WIP and how it can be interpreted. WIP has at least two different meanings:

Work in pro*gress*
Work in pro*cess*

Both of these meanings are widely used in the Lean literature. We happened to pick up "in process" from the literature we read as we learned about Lean and kanban. Throughout this book we're using *work in process*, but you can exchange it for *in progress* if you like.

5.1.1 What is work in process?

Work in process means all the work that you have going on right now. That includes work you're actively working on right now, work items waiting to be verified or deployed, and also the work sitting in your inbox that you haven't started yet: all the unfinished things you need to do in order to deliver value to the end customer.

WORK IN PROCESS AND LEAD TIME

Limiting WIP is one of the core kanban principles. It doesn't mean you should do less work, but that you should do less work *at the same time*. Limiting your WIP will help you complete more work in total more quickly.

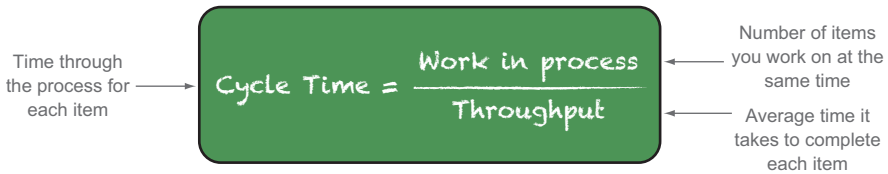
If you remember, back in the introduction we played a little game with the Kanbaneros—Pass the Pennies (see chapter 13 for details on how to run the game). The objective of this game was to show how different amounts of WIP affect your *lead time*, the time it takes for an item to go through your complete process. When the Kanbaneros were asked to flip 20 coins each before continuing, the total lead time was high; the WIP was 20 items at this stage. When the WIP was lowered, or limited, first to five (each worker flipped five coins and passed them on) and then to one (each worker flipped one coin and passed it on), the lead time went down—way down. The simple game showed you that the bigger the batches, the more WIP you take on, and the longer lead times will be.

A WORD FROM THE COACH If you want some more concrete examples of what a lot of WIP can mean before moving on, we can give you one from when we were writing this book. At one point we grew impatient and started to write a lot of chapters simultaneously, before finishing the ones we were working on. We had about eight chapters going at the same time, and some of them were closing in on being done. We then decided to restructure the table of contents and ended up moving a lot of stuff around. That change was considerably harder to do, took more time, and caused more pain with a WIP of eight chapters than, for example, two chapters.

This relationship between WIP and lead time has been expressed as a law in mathematics—in queuing theory, to be more precise. It's called Little's law; let's take a closer look at it.

LITTLE'S LAW

When talking about limiting WIP, Little's law often comes up. The law is a mathematical proof by John D.C. Little that says that the more things you have going at the same time, the longer each thing will take. It's a formula that looks like this:



As always, the first time you see something like that, the correct response is "Eh … what?!" If you're anything like us, it didn't make you go "Aha!" Let's throw in some real numbers and see if that makes it clearer.

Imagine that that your team takes on 12 items for a month and works on them all at the same time, resulting in a WIP of 12 items. They also typically finish 12 items per month, giving them a throughput of 12 per month. It's now trivial to calculate the cycle time for each item: 1 month.

$$1 \text{ month} = \frac{12 \text{ items at the same time}}{12 \text{ items / month}}$$

Let's do an experiment and—without changing anything about the way people work, the number of people on the team, or the work items—work with six items at the same time. That gives you a WIP of 6, still a throughput rate of 12 per month, and a cycle time of half a month. Wow! That's quite an improvement just by doing less stuff at the same time; don't you agree?

$$1/2 \text{ month} = \frac{6 \text{ items at the same time}}{12 \text{ items / month}}$$

For completeness—the reverse is also true. Double the number of items you work on, and the cycle time is doubled to a whopping two months. This is provided that the other conditions (way of working, item size, people on the team, and so on) remain the same.

$$2 \text{ months} = \frac{24 \text{ items at the same time}}{12 \text{ items / month}}$$

By handling fewer items at the same item (or limiting your WIP), you're lowering the cycle time and moving stuff faster through the process—without changing anything else! You can *probably* do it now and have your work items flow much faster through your workflow. This in turn gives you feedback faster and helps you to learn

about your process faster, which in turn gives you an opportunity to improve your process to move even more quickly.

WARNING Did you notice the small "probably" there? If your WIP is already pretty low, chances are you can't lower it and expect the other variables in Little's equation to stay stable. It might prove difficult to parallelize the work and have two people working on a task with the same efficiency. Decreasing WIP can have a negative impact on the average completion rate.

This isn't necessarily a bad thing; in fact, it's one of the strengths of kanban, because it will pose an improvement challenge. What do you have to change in your way of working to keep up your completion rate, even though you lower the amount of WIP? You can read more about this in section 11.1.2.

You now know a little about the concept of work in process and the theory of why you want to limit it. Let's be a little more concrete and take a look at how WIP can manifest itself in the software development industry.

5.1.2 What is work in process for software development?

In chapter 1, the coaches helped Team Kanbaneros understand what their WIP was. WIP in Lean manufacturing is obvious because it often manifests itself physically: items piling up in front of a machine waiting to be processed or finished items on the floor waiting to be moved to the next step in the processing chain. Once you learn that it can have a negative impact on your work, it's pretty easy to spot, because it often involves physical things.

WIP in knowledge work isn't visual, and this is the driving force behind wanting to visualize work, to make work apparent and obvious where it previously wasn't. This is one of the main reasons to create boards and stickies that show your work items and their status, as described in chapter 3. The work is there even if you don't use a board; you just don't see it as clearly.

But there's more to WIP than the number of items you're working on at the same time. Considering only the *number* of things you're doing at the same time is a simplification. In this section, we'll look at a couple of ways that WIP can manifest itself in the world of software development.[1]

SPECIFICATIONS NOT BEING IMPLEMENTED YET

If you think about it, specifications have a "best-before" date. You could say that they rot if they're left lying around.

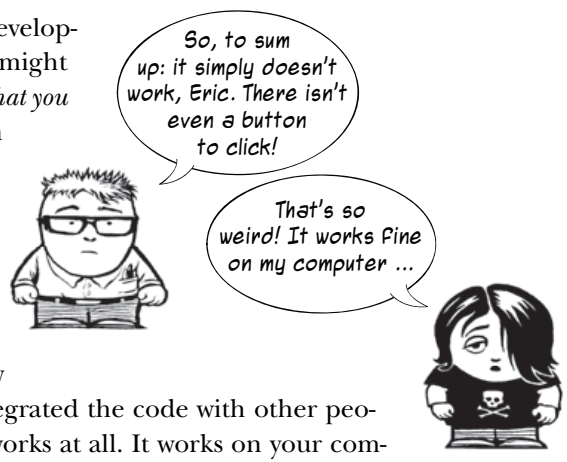
[1] See Mary and Tom Poppendieck, *Implementing Lean Software Development* (Addison Wesley Professional, 2006, http://amzn.com/0321437381), for more on this.

Imagine that today you write a specification, and then you leave it. What are the odds that anyone could pick it up in six months and start coding from it? Even after only two months, there's only a slim chance. Things in the business environment change, and the system for which the specification was written has probably also changed during the waiting time. At the very least, you have to go through the specification again to be sure it's still valid.

A specification that is written and lying around waiting to be implemented is work in your process.

CODE THAT ISN'T INTEGRATED

Continuing along a stereotypical development process, the next thing that might increase your WIP is *implemented code that you haven't checked in and integrated* with other people's code yet. That's also WIP, because you don't know how much work there is still to do before you're done with the work item.

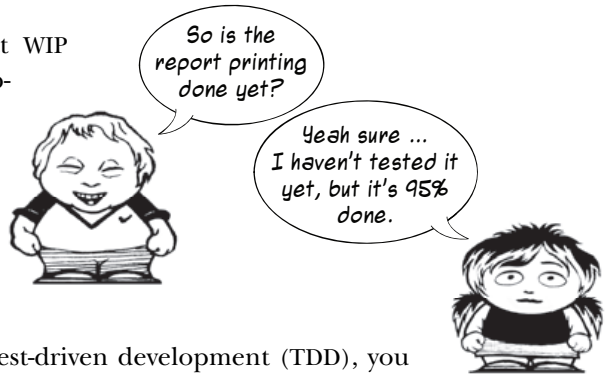


If you've ever heard the phrase "It works on my computer," you know what we mean. If you haven't yet integrated the code with other people's work, you still don't know if it works at all. It works on your computer, with your settings, and in your environment.

Checking in and integrating your code often is a good way to not accumulate loads of integration work and to get quick feedback on the quality of the work you have done so far.

UNTESTED CODE

Untested code is another way that WIP manifests itself in software development. To write code without having a quick way of finding out if it works or not is an excellent way to build up a stock of unfinished work.



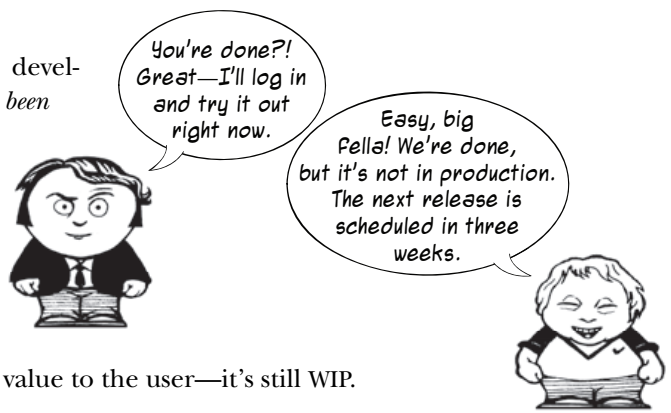
Automated testing is one way to handle this problem. By using automated unit testing or test-driven development (TDD), you get quick feedback that you're not introducing defects into the existing software. By doing automated acceptance testing or specification by example, you get feedback that you're building the right application.

A WORD FROM THE COACH Test-driven development (TDD) is a design and development practice in which you start by writing a small test for the code you're about to write. It's a micro-specification for the next little chunk of code you need to fulfill your task. As a side effect of working this way, you get a suite of test cases for all of your code. TDD is all about developing *things right*!

ANOTHER WORD FROM THE COACH Specification by example is also known as behavior-driven development (BDD) and is a powerful way to, in essence, write your specifications as executable test cases. Specification by example is about communication and making sure everyone understands each other. Doing this badly, in our experience, takes a lot of time, because you then have to go back and forth and anchor the information around the feature to build. By using concrete examples early in the process, as you specify the functionality, you increase the likelihood that everyone means the same thing when you talk about the feature. In essence, specification by example is all about developing the *right thing*.

CODE NOT IN PRODUCTION

Finally, code that has been developed and tested but *has not been taken into production* is also WIP. You still have some work to do, and you still don't know if the feature is functioning or if it will have a bad effect on other parts of your system. Above all, it's not yet providing any value to the user—it's still WIP.



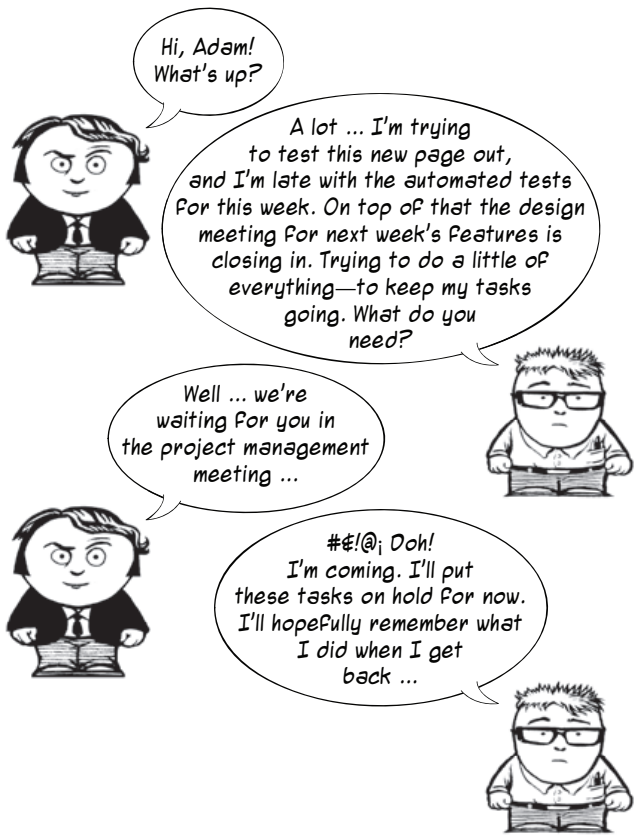
If you think about it for a while, code in production might also be WIP. The work isn't necessarily "done" just because it's in production. Having the code in production, working, isn't what matters—it's how the users receive it, whether they benefit from it, and whether your software accomplished the (business) impact that you intended. If the user behavior isn't affected, are you really done?

We have now examined what WIP is and how it manifests itself in our industry. Let's dive in and look into the effects of too much WIP.

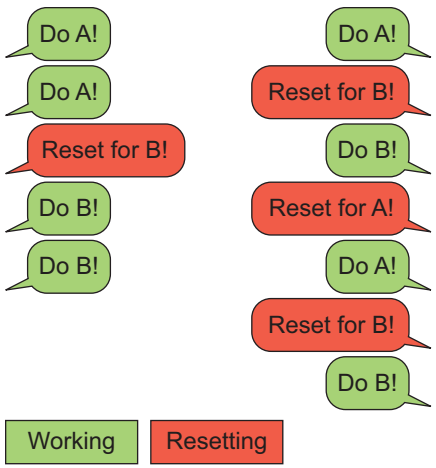
5.2 Effects of too much WIP

You might now think, well, a lot of work in process is a bad thing, I know. But how bad can it be? What happens if you have too much WIP? This section examines the answer in detail. It's demoralizing reading, but keep your head high. You already know how to tackle this: limit your WIP!

5.2.1 Context switching



If you were Adam in the preceding cartoon, do you think you'd have to struggle to remember what you were doing when you came back? You probably would, right? This happens due to something often referred to as *context switching*.

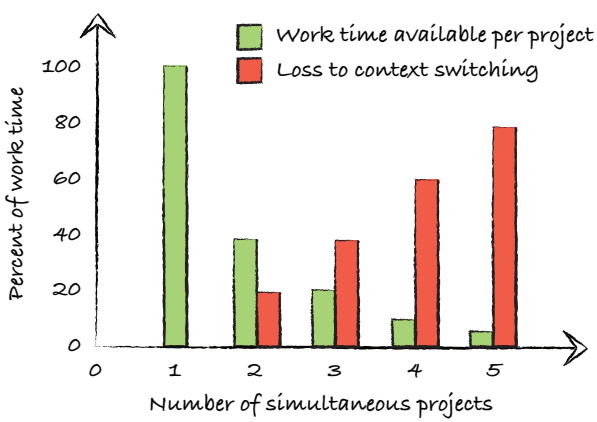


Context switching was originally a concept from computing that describes how the computer stores and retrieves state for one process as it switches to another. This is the basis of multitasking several processes on one CPU. The feeling of "trying to get back to whatever I was doing" is the same thing, but for humans. When switching between several tasks, you need to set up in your mind the "state" of the task you were doing earlier. Think of it as human context switching.

You can compare this with the setup time for a machine in a factory. If the machine is rigged to produce Model A, it will take some time to readjust it to produce Model B. It's then easy to understand that if every other piece is alternating between Model A and Model B, a lot of time will be lost due to setup.

That's exactly what you experience when doing context switching in knowledge work. You lose time and focus for every task you're trying to keep in your head at the same time.

One study[2] showed that as much as 10% of your working time per project is lost to context switching. This means if you're running two things at once, you have only 40% of your available working time per project to spend. With five tasks going at the same time, you have only 5% per project left.



[2] Gerald Weinberg's book *Quality Software Management: Systems Thinking* (Dorset House, 1991, http://amzn.com/0932633226) is often referenced as the source of these numbers, although we haven't been able to find them there. But Weinberg has confirmed that he's the source in private correspondence with us.

Interestingly, another study also showed that context switching represented a 10-point drop in IQ. That's more than twice the number found in studies on the impact of smoking marijuana (!). So if you have to choose between the two, you know what to do.[3]

What can you do, then? These projects and tasks are part of your WIP—the WIP for you. And by now you know what to do with WIP: limit it! Keep it to a minimum. If you have to keep several tasks going, at least try to have as few as possible at once. Try to complete one before starting another. Doing this will help you avoid context switching and complete each task more quickly (according to Little's law).

Now you know something about the problem with context switching in our type of work. Let's continue and take a look at another of the problems that a lot of WIP causes: more work.

5.2.2 Delay causes extra work

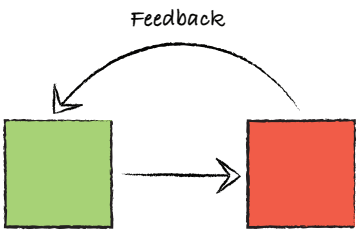
Imagine that you happen to introduce a bug in the application you're writing. If you're notified about it immediately, within minutes, it's simple to fix, and nothing bad will have happened.

Now consider the same situation, but suppose it takes two months before somebody finds the bug and notifies you. It's now considerably more work to fix the bug. You first have to remember what the feature is, what you did about it, and how to fix it.

[3] We want to be clear that the last part was meant as a joke. We wouldn't want anyone to think that we're proselytizing for using marijuana. Marcus, a member of the Salvation Army, particularly wants to underline this. The study is no joke, though: "'Infomania' worse than marijuana," BBC News, http://mng.bz/HjcX.

Then you might have to set up the system in the state it was in at the time to be able to reproduce the bug. Finally, the rest of the system might well have changed since then, making the bug much harder and more complicated to fix. And that is only considering you and your situation. You might have to involve others to be able to reproduce the defect (testers, system admins, or DBAs to set up the system in the correct state, for example).[4]

All this extra work is caused because of the *delay* from the time you introduced the bug until when you were notified of its existence and could do something about it. You get more work (in process) due to the delay itself.

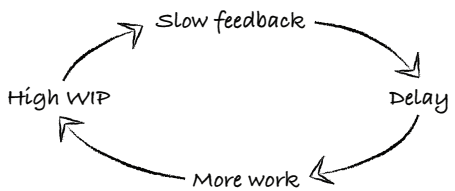


This problem boils down to the feedback loop being longer. Not only does a lot of work going on at the same time make each item go more slowly, but feedback on how each work item was received and behaves in production takes a longer time to get, too.

And that is a bad thing.

Feedback is an essential part of every agile process. Feedback is the creator of knowledge. It tells you about the quality of your work and the quality of your workflow. What works? What should you change? What shouldn't you change?

The more quickly you can get feedback, the more quickly you can change a bad process into a slightly better one. So you want to fail fast if there are any problems. Delayed feedback makes it difficult to connect the effect to its cause, making learning very difficult or even impossible.

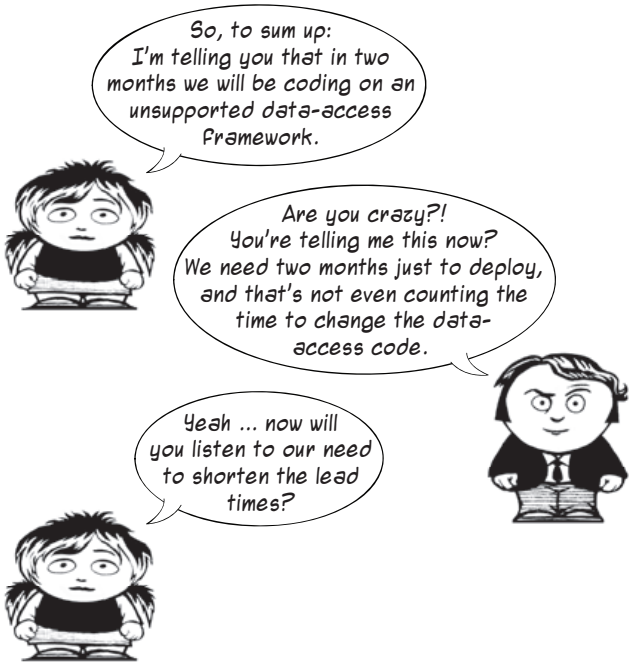


As you've seen, higher WIP causes feedback to become slower and slower. And the slow feedback itself causes even more work to be created, which causes even higher WIP, which prolongs the feedback loop, and … well, you see where this is heading, don't you? By lowering your WIP, you can stay on top of this problem and get feedback quickly on the small work items that move rapidly through the complete workflow.

But more WIP, longer lead times, and prolonged feedback loops: are those such as big deal? Yes, and the next few sections will show you examples of how they can haunt you.

[4] At one client, Marcus had a meeting cut 30 minutes short because the other person had to read up on a bug they were about to discuss in the next meeting. "I filed it one and a half years ago" was the understandable reason for her need to go and read up on what she wrote back then.

5.2.3 Increased risk



With more work going on at the same time, you're increasing risk. This has to do with not being able to change quickly and hence be more sensitive to changes in the world around you.

For example, let's say that you have a process with long lead times. If your customer requests a change in a feature, it will take a long time to build and take into production. During that time, the feature may become obsolete, or a competitor may pick up that request and implement it first.

An example of such a feature could be integration with social platforms such as Twitter and Facebook. Many online services now give you the ability to log in using credentials from these platforms. If you have a long lead time, say a year, someone else might have already created a copy of your service with the social media integration in place.

Other places where the ability to change fast is important are within technology, as well as in the environment in which your company operates (trends, laws, and other regulations, for example). If you can't change fast and get new features or changes to your customers fast, you stand the risk of losing relevance for your customers, your service becoming obsolete, or even being outmaneuvered by someone else.

5.2.4 More overhead

A really nasty thing about a lot of WIP is that more work will be created from the need to coordinate all that work. It's a vicious circle that quickly can spin out of control and in which you end up only doing coordination. This resembles the situations we described in section 5.2.2, in which we talked about delays creating more work. The delay we talk about here is that you have longer lead times with a lot of WIP.

Imagine that you're taking part in three different projects at once. Not only does that mean a lot of context switching for you, but those involved in the three projects will also likely ask you to do reporting, time tracking, and planning for all three projects. Because you're moving in and out of the projects, you probably need to do a lot of hand-overs and coordination, too. Much of this extra work is created because you're doing several projects at the same time. If you were assigned one project only, you wouldn't be asked to do a lot of the tasks required to coordinate your work among three projects.

This kind of situation often occurs in organizations with a strong focus on using "resources"[5] to the full. If you instead have a focus on keeping lead time short for the things you produce and smoothly flowing items to the customer, a lot of these extra tasks will be considered wasteful, and you will strive to eliminate them.

[5] Often the word *resource* means people in these organizations, too.

A WORD FROM THE COACH We have both worked as contractors for quite some time. In this business, time reporting is of the essence, because that is our product: as contractors, we're billing the customer for hours. Without fail, the time-reporting system in contracting companies is typically a mess, is complained about, and takes a lot of time to use. More often than not, you'll hear this at any contracting firm: "So who should we bill for the time it takes me to do the time reporting?"

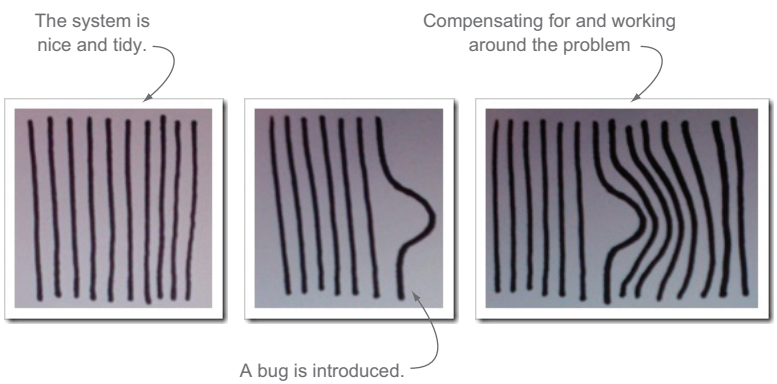
At Marcus's employer, Aptitud, he's tried to resolve the problem of this wasted time by reporting deviations from a norm. If he's assigned to one project for 60% of his time, and that's the number of hours he worked for that client, he does nothing and Aptitud will bill the client for 60% of his time. If he, for some reason, is two hours short of 60%, then he reports those two hours as a deviation.

5.2.5 Lower quality

If you're a developer, the quality of your code may suffer from long lead times. This has to do with the prolonged feedback loop that lasts from the moment you write code until you know how it was received and how it functions in production.

Imagine that you happen to cause a bug to be introduced; you assumed that the customer had one name, but there should be both a first and a last name. The error passes unnoticed, because all the customers in your system up to that point were companies and used a single name field.

Then one day, several months later, your first "private" customer tries to log in and gets upset because he can't enter his first and last names as he is used to doing. Correcting this bug isn't easy; a lot of code that depends on customers having a single name property has already been written. You have worked around the bug without noticing.



Scott Bellware[6] once compared this situation to introducing a hernia in the system, and he illustrated that with the preceding pictures. At the outset, your system is nice and tidy. When you introduce the bug, one line gets distorted. Until you're informed about the bug, you'll work around the problem, creating bad code that compensates for the bug in the system. It's not until you fix it that you can get back on track and have a well-aligned system again.

Compare that situation to one in which you're immediately notified about the bug you introduced and are able to fix it within hours of writing it. No hernia, no fuss: simple and quick.

The difference between these two situations is lead time: lead time from when you introduced the bug until you're informed of it. During that time, your code has suffered in quality, which means fixing the bug will take longer and will be more difficult.

5.2.6 Decreased motivation

Long lead times can decrease the motivation of your team, too. It's not hard to see why, because stuff ending up in another queue isn't a motivation booster.

[6] See Scott Bellware, "Design Flaws, Hernias, and Anemic Quality," February 9, 2009, http://mng.bz/f2x1, for more.

Take Daphne in the preceding cartoon. She has been slaving away on that feature that Adam has asked her to complete by today. It even took her part of the night, but she knew that it needed to be ready for Adam. He demanded it, because the last three times he had to wait on her. Making Adam happy this time provided the sense of urgency (or motivation) that Daphne needed to go the extra mile and finish the item today.

But when Daphne arrived in the morning, she found out that the item was the last in a long array of work items that needed to be tested. Her motivation to deliver on time took a severe hit.

In this example, it was Adam who had a lot of things going on at the same time. Because of that, he couldn't give Daphne feedback on her work until two weeks later. That in turn resulted in Daphne's motivation dropping; and when Adam finally came back, she couldn't have cared less about the testing result. He was interrupting her at that point, because she had moved on to other work.

What should they do about that? Well, herein lies the real reason to strive for lower WIP and hence shorter lead times. It will expose problems[7] for you: problems that, if you try to fix them, will make your flow even faster, even smoother. Sadly, kanban doesn't say much about how to fix these problems. You have to find out for yourself by seizing one unrealized improvement opportunity at a time. We have devoted chapter 10 to practices that can help you find and implement process improvements.

5.3 Summary

This chapter talked about work in process and specifically discussed limiting WIP in order to get shorter lead times:

- WIP is a common abbreviation, and it has at least two meanings: work in pro*gress* and work in pro*cess*. We prefer *work in process* and use that throughout the book.
- Little's law tells us with mathematical certainty that the more WIP, the longer the cycle time for each work item will be. You should limit the WIP to get faster flow and shorter lead times.
- WIP can take many forms, and we looked at some common ones in software development:
  - Specifications that haven't been implemented yet
  - Non-integrated code that "works on my computer"
  - Untested code, which may or may not live up to your standards
  - Code not in production, which sits and wait for the next release
- Problems and bad things can follow from having a lot of WIP:
  - More context switching
  - Prolonged feedback loop, which in turn causes extra work for you

[7] Sorry—"unrealized improvement opportunities" is much better.

- And we mentioned some ways that WIP can manifest itself:
  - Increased risk
  - More overhead
  - Lower quality
  - Less motivation

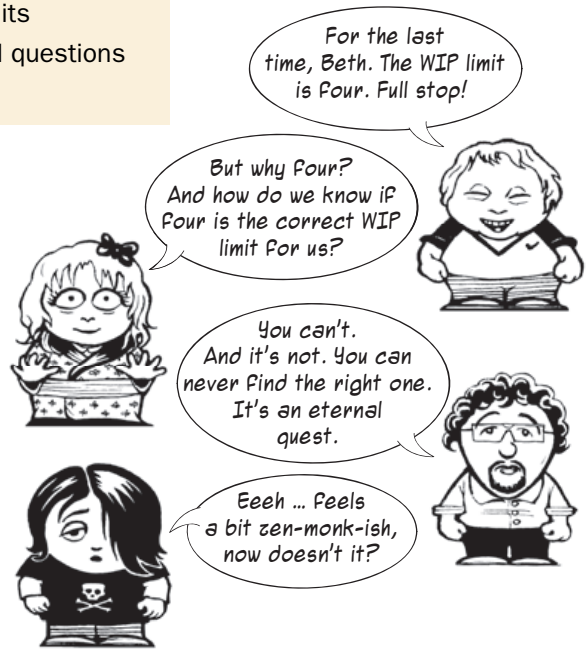
You now have solid knowledge of what work in process is and why too much of it is a bad thing, and you have a theoretical understanding of what to do about too much WIP. The next chapter will be much more practical, and you'll start setting WIP limits for your team.

6 Limiting work in process

This chapter covers

- Principles for finding suitable WIP limits
- Common ways that teams limit WIP
- Common ways to visualize WIP limits
- Answers to some frequently asked questions regarding WIP limits

From the previous chapter, we hope we have now convinced you that a lot of good things come from limiting your work in process (WIP). By now you're probably full of questions: What is the right limit for you? How do you go about finding the WIP limit? Are there some good starting points that can guide you?



As you soon will find, there are no hard rules here. Finding a suitable WIP limit is not only contextual and dependent on what you want to achieve, but also like hitting a moving target. The WIP limits can and should change. At the outset of this chapter, we can already reveal a secret: *the goal is not to limit WIP.* WIP limits are only a means to drive you to improve. Improve to achieve a better flow, which is the theme of the next chapter.

That's why you'll see a lot of "It depends" and "You can do this, that, or the other" in this chapter. It's not a sign of us not knowing; it's that you need to find what works best for you and your team.

No need to worry, though; we have packed this chapter full of guidance and principles for finding a suitable WIP limit, as well as practical ways many teams go about visualizing and setting their WIP limits. Let's dive in and see how you can reason your way to finding a WIP limit for your team.

6.1 The search for WIP limits

Searching for a suitable WIP limit for your team can be tricky business. What is right for you and your team right now? The answer is the old-favorite answer of consultants worldwide: "It depends."

And it does. It depends on

- How much pressure there is to continuously improve the organization
- The number of people on the team and their availability
- The shape and size of the work items you're working with

And the list goes on.

To find a WIP limit for your team, here are few basic rules of thumb, but be prepared to change the WIP limits as you learn more:

- Lower is better than higher.
- People idle or work idle.
- No limits is not the answer.

Let's examine these approaches in detail.

6.1.1 Lower is better than higher

A lower WIP limit is generally better than a higher one because you want to limit the number of items you work on as much as possible. This will give you better lead times and faster feedback and force you to remove impediments. These are all things that help you improve the flow of work items in your process.

Setting the WIP limit *too* low, however, could make your process come to a grinding halt, because it will quickly bring out any problems in the process. Imagine that your entire team is working on a single work item, and all of a sudden you need an answer from a customer in order to continue. Because that work item is the only thing you were doing, you have come to a complete stop.

Problems are symptoms of things that will have to be changed in your process to improve your flow. Finding many problems can quickly become a painful experience

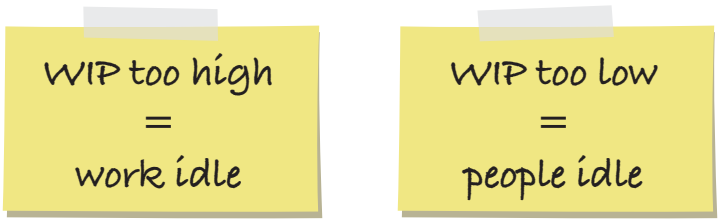
for teams that aren't equipped to handle a lot of issues like these at once. Instead of feeling encouraged to improve and solve their problems, a common reaction is to give up and stop caring that they're breaking the WIP limit on a regular basis.

You want to search for a low WIP limit, but not too low. There are signals to guide you, as you'll see in the next section.

6.1.2 People idle or work idle

Tuning the WIP limit can easily become something that is talked about forever: striving to find the one true limit can result in a lot of discussions in which no one is able to decide what is best. This isn't time well spent; it's better to try something and then adjust.

A simple method to help you adjust the WIP limit up or down is the following:



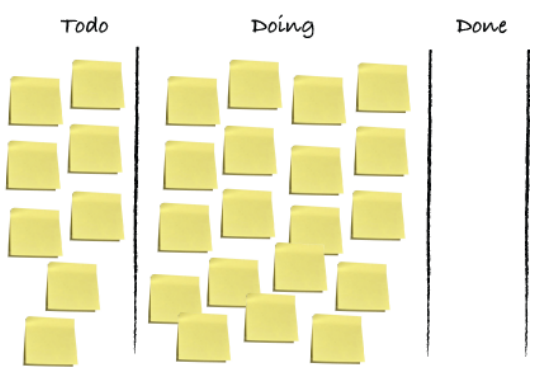
If your *WIP limit is too high, work will become idle.* There will be work items that no one is responsible for. This might be a good time to lower the WIP limit.

With a *WIP limit that is too low, people will become idle.* All items are being worked on, and there are people without work. You can now collaborate to get items done or raise the WIP limit.

6.1.3 No limits is not the answer

Be sure not to fall into the trap of not setting a WIP limit at all! This is one of the most common mistakes we see in teams starting with kanban. Kanban involves three simple principles; don't remove one of them too soon.

The risk is that you'll end up with a board flooded with work, as you see in the figure to the right, showing your inefficiencies and your failure to move work through your process. Eventually, the board will end up not being used, because you can't see the work items for all the work on the board.



Office spaces around the world have unused boards covered with

stickies. Don't add your board to that sad bunch; instead, make the board a tool that helps you improve.

Also, as we'll discuss further in chapter 7, removing the WIP limit will remove your incentive to improve. With no WIP limit, nothing pushes people to do better. If you allow an unlimited number of items on your board, nothing will force you to complete the ones on there before starting new work.

This is because (if you remember from chapter 5 about work in process) the more stuff you have going on at the same time, the longer the lead time will be for all the work in your process. You want a constraint that pushes you toward finishing work and toward improvements in your process. We'll dive much deeper into this in chapter 7.

Let's turn our attention to some principles of how to decide on a suitable WIP limit.

6.2 Principles for setting limits

Finding your WIP limit depends on your context and particular situation. In this section, we'll take a look at some common ways that teams we've coached and heard about have agreed on their limits.

Remember that it's not so much about finding *the* WIP limit, but rather, visualizing your current WIP limit; it's the best so far. You *should* change it as often as needed. Embrace change; it's good for you.

As always, use the examples here as inspiration, and expand on them to suit your needs. Let's start with a simple principle, one that might already be stuck in your head from previous chapters: stop starting, start finishing.

6.2.1 Stop starting, start finishing

A simple yet powerful starting point to limit your work in process is agreeing to *strive to complete current work before starting anything new.* This will keep your WIP low, because you only start new work when something else is finished.

An easy way to begin doing this is to adopt this slogan for your team:

Yes, you have to think hard when saying it the first few times, but then it sticks. A good idea might be to put this statement above your board, where you'll meet each morning; or why not use it as the opening or closing phrase of your daily standups?

Hill Street Blues—roll call

To beat that sound bite into your head, you could use the *Hill Street Blues* roll-call ending. As people leave the standup, say:

"Oh, one more thing. Stop starting, start finishing, everyone."

Wait a second. And then add the famous ending:

"And let's be careful out there."

By using a reference to *Hill Street Blues*, we have probably dated ourselves pretty well. It was a police series in the 1980s that was a big hit in Sweden. Every person who has seen it remembers the ending of the morning roll calls at the police station; everyone rose, and the sergeant called out, "One more thing … let's be careful out there."

There is great power in this simple slogan, because it pushes you to a lower WIP. It also sums up a basic pattern of kanban: to prefer to finish work before starting something new. Doing so will help you not only to limit the number of work items you work on at the same time, but also to shorten lead times, take work through the whole value chain, get feedback, and improve your process.

Let's finish this section with an inspirational quote to help you remember what's important:

> *It's not "the more you start, the more you finish," it's "the more you finish, the more you finish."*
>
> —David P. Joyce

Soon you may need a bit more control over your WIP limit, and at that point this principle might be too coarse grained. But it's a great start to emphasize a principle.

6.2.2 One is not the answer

Striving for a low WIP limit is desirable because it gives you better flow (see chapter 7), but setting it too low will probably be too disruptive for your flow. For example, if you have a WIP limit of one, any disturbance in the flow—such as handoffs, people out of the office, and so on—will cause all work to stop. Most organizations aren't ready to handle all those improvement opportunities (you remember that's what we call problems, right?) at once, so you probably should go with a WIP limit higher than one item.

Mob programming

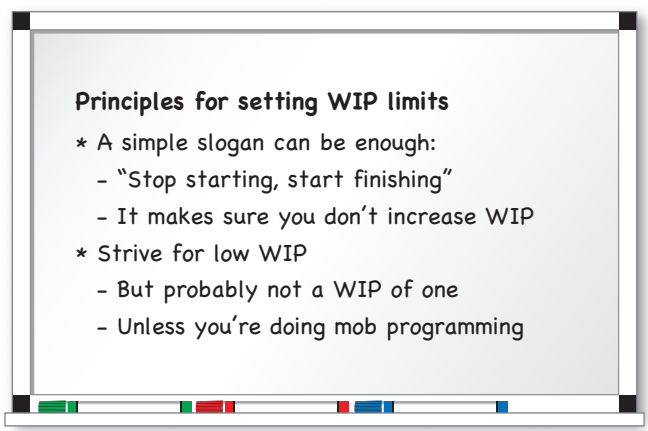
Mob programming is an interesting and novel approach to system development pioneered by a team coached by Woody Zuill. The basic concept of mob programming is simple: the entire team works as a team together on one task at a time. That means: one team, one (active) keyboard, one screen (projector of course). It's just like doing full-team pair programming.

A team doing mob programming has a WIP limit of one. That's very effective when it comes to completing the work being worked on as fast as possible. Any question, any problem, will be tackled immediately; and all the people needed to solve the problem are present in the room and have time to solve any issues right away.

But this approach is very inefficient when it comes to utilizing the people on the team to the fullest. Not every person on the team is actively typing code.

The question is, are you selling keystrokes or completed features?

For many organizations, mob programming is too extreme; a WIP limit of one is probably not suitable for everyone. But for the organizations that can handle it, work flows very fast indeed.



Summing up this section on guidance for setting your WIP limits, we can say that in most cases, setting the WIP limit to one isn't a good idea. You should strive for a low number, but not necessarily one.

By now you're probably longing for some practical advice on how to set WIP limits using common practices, visualizations, and methods that other teams have found helpful. Well, let's get to it. You'll start by setting a single WIP limit for the entire team.

6.3 Whole board, whole team approach

In this section, we'll take a look at a couple of ways to limit the total number of work items on the board for the whole team. This can be a simple way to get started and may be all that's needed for your team.

Remember that there is no true WIP limit to be found. WIP limits are tools that you use to improve. You might ask questions like "Will this WIP limit help our work to flow better or make our flow worse? Will this WIP limit render a lot of people idle? Will a lot of work be idle? Is that a problem, and what can we do to resolve it?"

6.3.1 Take one! Take two!

Many teams that we have coached have gone through a similar journey to reach a suitable WIP limit. Their experience illustrates the kind of reasoning that we think is necessary concerning WIP limits, and we want to relate that reasoning here. Learn from this, and then pick a limit that suits you.

Let's assume you have a five-person team. You could, for example, start out boldly and decide on a WIP limit of one per person, making the total WIP limit equal to the number of people on the team.

Pretty soon, you'll find out that this approach creates some problems; if you become blocked, you can't pick up new work without breaking the WIP limit, for example. You may decide it seems more reasonable to allow for a maximum of *two* items each, so people have something to do if one item is blocked. That makes the total WIP limit equal to the team size times two.

But that setup might not be reasonable either, because it allows for situations in which every person on the team can be blocked on one item and still keep working on another without breaking the WIP limit. That's a perfectly valid situation with a WIP limit setup like that, but in many instances this isn't pushing the team enough. You

want the WIP limits to push you into resolving problems, not allow you to work without noticing them.

To push the team, you can back off from a WIP limit of 10 items and decide on maybe 8. If all team members are now blocked on one item, you'll start to break the WIP limit if everyone picks up another one to work with. You get a gentle nudge to do something about the blockage, because you exceed the WIP limit earlier with 8 items than you would with 10 items. If you want to get this warning even earlier, lower the WIP limit a bit more.

Note that the numbers in our examples aren't that important, but the *reasoning* (what happens if you're all blocked, will this improve flow, and so on) to find your WIP limits is. Remember—there is no right WIP limit that can be calculated. You must find the one that suits your team and your situation the best. The most rational thing is to try a WIP limit and then change it as needed.

Also, think about what kind of behavior you want the WIP limit to drive. It's the mechanism that drives improvements.

6.3.2 Come together

The previous section allowed every person to have at least one item going on at the same time. If your team is using collaboration practices like pair programming, you should probably lower the WIP limit even more. In such cases, you could set the WIP limit to a number lower than the number of people in the team. You could take the number of people on the team *divided* by 1.5 (or some other number you see fit) to yield a low WIP limit.

For our example five-person team, this will add up to roughly three items total to work on.

With a WIP limit of three for a five-person team, people will quickly end up with nothing to do, if everyone works alone. To be able to handle a WIP limit of three and still engage people in the process, the team members will have to cooperate on items to

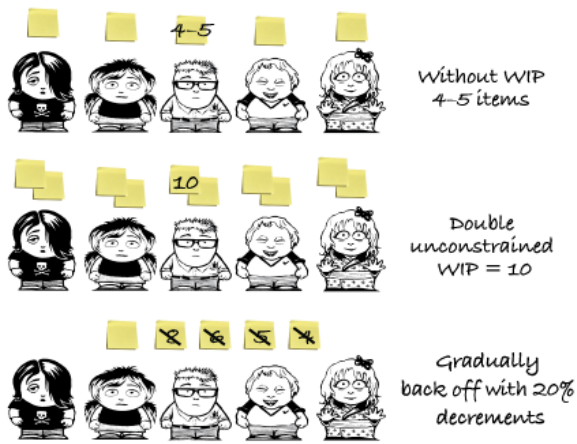
finish them by using pair programming, writing the specifications together, or testing in pairs: exactly the kind of collaboration that agile teams seek. As a nice side effect, the work items will move fast across the board.

This might be a big change for some teams that aren't used to collaborating closely. As always, don't set the WIP limit too low to start, but rather lower it little by little and improve your process at the same time.

6.3.3 Drop down and give me 20

One technique to set a suitable WIP limit was introduced by Don Reinertsen.[1] He proposed that you first observe what a normal level is in an unconstrained system. That means: what is the normal load of work items in your system? Start where you are: visualize the work items, and count them to learn what your current WIP is.

Then double that amount of WIP and use that number as the limit, allowing for twice as much WIP as you have normally. You double the WIP limit because it can be statistically deduced that with normal variation, almost no teams will reach the limit. It's safe to say that this limit will have no effect at all to start with—everyone should be able to pull it off. From this new limit, you can then incrementally go down by 20–30% decrements—say, every month or every two weeks—until you start to experience problems, queues build up, or you see people idle between tasks.



Let's see this in practice. Say that a team usually has roughly four to five items going on at once. Following Mr. Reinertsen's reasoning, you should start by allowing for 10 items on the board. You then back off that number by 20% at regular intervals. The first iteration you're down to 8 items, and then 6, and then 5, and so on.

As you do this, you're not only lowering the WIP limit, which will flow items through your system faster. You're also starting a trend of continuous striving to improve: a mindset that characterizes Lean thinking.

This approach puts the focus on an important aspect of WIP limits; they're cheap to implement. You pick the number and don't take on more work than that. If that doesn't work out for you, if too many work items are idle, you can go back up to the level you had before. This makes WIP limits excellent for use in experiments.

[1] Presented at a keynote at the Lean Kanban Central Europe conference 2012. See www.lean-kanban.eu/sessions/reinertsen/ for more information.

What happens if you lower the limit by 20%? Try it and see; if it doesn't work out, go back and think about what happened. What would you have to do differently to be able to lower the WIP and still keep items flowing through your system?

This approach teaches the team to continuously evaluate the limit to improve the flow. Because the initial WIP limit is high, decreasing it the first couple of times won't cause any big problems. This helps the team get into the habit of evaluating and changing their limits regularly.

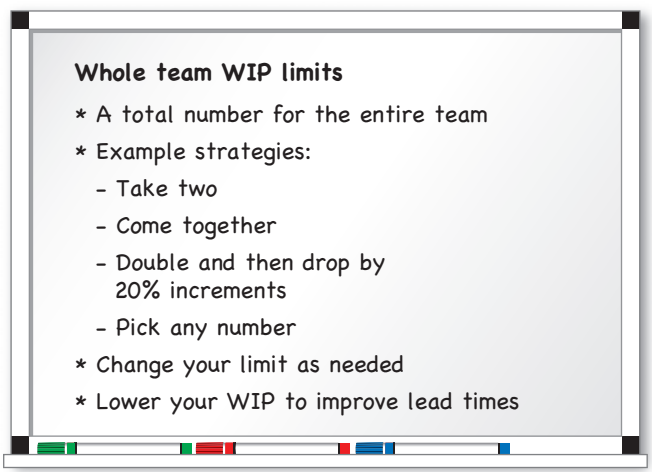
Although we used a whole-team WIP limit approach in this example, it makes perfect sense to apply the same thinking with a column-based WIP limit approach (as discussed in section 6.4). You could stop decreasing the limits for the columns where you feel pain and keep decreasing them for other columns.

6.3.4 Pick a number, and dance

If you don't know what a suitable WIP limit could be, pick a number out of the blue. Yes—you read it correctly. Pick any old number!

Or rather, make an educated guess of what could be smart, but don't think too much. This number, as we've said many times, could and should be changed as you progress and as you see fit.

Use the principles for finding a good limit, as described in section 6.1, to guide you toward setting your WIP limit.



Until now we have considered WIP limits for the whole board and whole team. Doing this, you don't concern yourself with how the work is distributed among the steps in your workflow (the columns on your board). Setting one single WIP limit, like the examples we've considered up to now, is sometimes referred to as a system with a constant WIP (also known as CONWIP).

Let's now take a look at some other approaches for how to limit WIP. Could you limit WIP for only parts of your workflow? Why would you do that? And how could that be visualized?

6.4 Limiting WIP based on columns

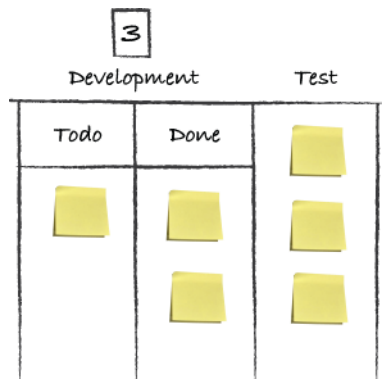
Most teams that we have worked with set WIP limits per column instead of for the entire team. This gives them a bit more fine-grained control over how their work flows and an opportunity to handle bottlenecks and uneven flow. That said, setting a WIP for the entire board could still be useful and, if you don't know where to start, provides a good starting point.

In this section, we'll dive deeper into common ways of limiting WIP per column, some things to consider when doing so, and what you can gain from a WIP-per-column approach.

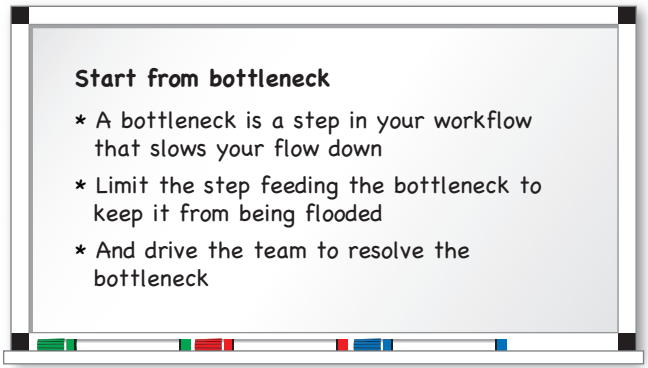
6.4.1 Start from the bottleneck

All workflows always have a bottleneck somewhere, which sets the pace for the entire workflow. Increasing throughput upstream from the bottleneck will just pile up work in a queue in front of it, and trying to increase throughput downstream from the bottleneck is futile because there won't be enough input to work on. If you can identify the bottleneck in your workflow, it makes sense to use WIP limits to help resolve it. For more on bottlenecks and how to discover and manage them, see section 7.5.

Let's look at an example, in the board to the right. Here we have a workflow where work items are piling up in a queue in front of a bottleneck: the Test column. If you do nothing, work will keep piling up, and WIP and lead times will increase. If you instead put a WIP limit of, say, 3 on the whole Development column, the developers will have to stop doing Development work when the queue starts to build. What should they do instead? They could help the testers finish their work. Whatever they do, they shouldn't pull in new work, because that will only build more WIP.



As you can see, putting a WIP limit on the step feeding the bottleneck will stop the bottleneck from being flooded and drive behavior to resolve the bottleneck, thus improving the entire flow.



6.4.2 Pick a column that will help you improve

Why did you buy this book? What is the reason you think kanban will help you improve? Most teams already have a pretty good idea about where (some of) their problems lie. A common scenario is to use kanban in order to collaborate together on fewer work items in order to complete them faster. In a team of mostly developers, it might make sense to put a challenging WIP limit on the Development column, where most of the active work is being done, in order to drive collaboration.

Many teams start with one of the "whole team, whole board" approaches described in section 6.3 and distribute numbers over different columns. You can also apply the total to just one column: for example, putting 1.5 times the number of developers as a WIP limit on the Development column.

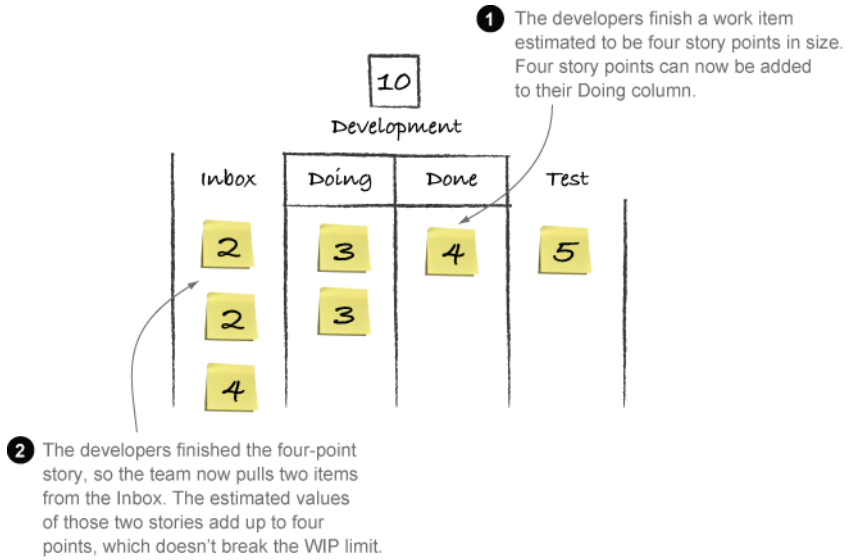
6.4.3 A limited story, please

There are as many ways to limit WIP as there are teams in the world, but we'll mention one that sometimes comes up as a question: can you limit your work in process based on estimated size of the work item—for example, by using *story points*?[2] (You can read more about story points in chapter 9.)

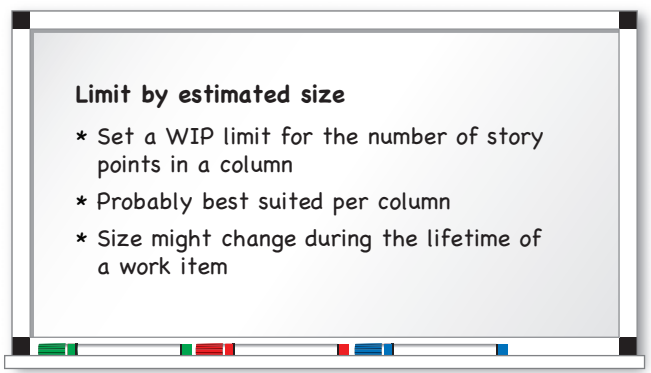
As we've said, you'll probably have to experiment to find a WIP limit that is suitable for your team. Start with your gut feeling and adjust the limit up or down, as described in section 6.1. This goes for estimated effort size of your work item (for example, in story points) as well as for work items. Many teams using Scrum do this already, because when they're planning their next iteration, they look at how many story points they typically finish and pull that amount of new work into the iteration.

[2] Story points are a way to estimate effort for a team, using relative measures; they're used in many agile methods such as Scrum and Extreme Programming (XP).

Limiting WIP by estimated size means you only pull new work as long as it keeps you under an agreed-on limit: for example, under 10 story points. The next figure shows an example of a simple board where the developers have a WIP limit of 10 story points.



This WIP-limiting approach is often more suitable to decide per column, rather than for an entire team, because the size of a work item may change the more you work with it. For example, when you start analyzing an item, you might find that it will take less effort to implement than you first anticipated.

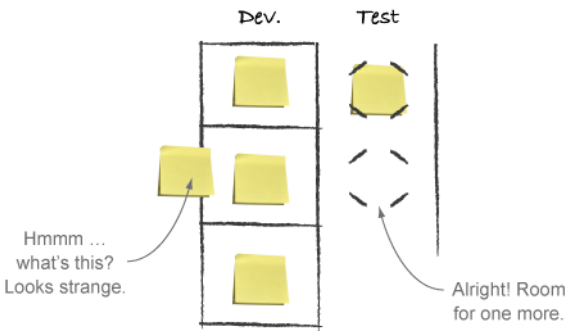


6.4.4 How to visualize WIP limits

Column-based WIP limits are often visualized by drawing them above each column of the board. But variants of this are plentiful.

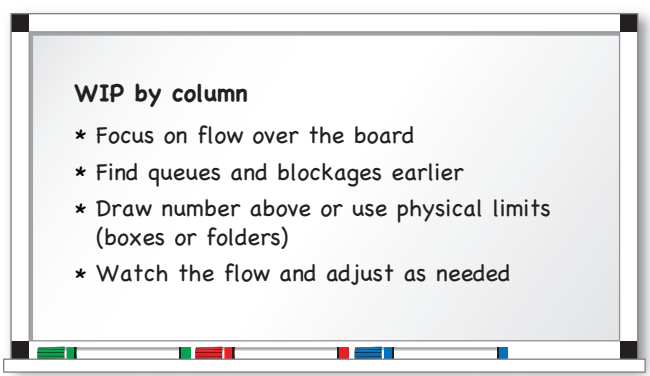
Some things we've seen while working with teams are as follows:

- *Boxes for each work item*—This will make it apparent when you break the limit, because you have to put your work item outside of a box or on top of another work item.
- *Plastic folders or other physical placeholders for each allowed work-item card*—This is similar to the drawn boxes and is also a good visual indicator that helps you to keep track of how you're doing against the established WIP limit.



What you use is up to your imagination and what suits your team. Don't be afraid to experiment. Try something new, and fail, instead of being content with a suboptimal setup.

As you can see, there are lots of ways to use WIP limits on the columns on your board to help you find problems, bottlenecks, and other things that slow down flow in your process. What will work best for you is left as an empirical exercise: experiment. Try something, and change to improve.



Let's now turn our attention to another way of limiting WIP: by person.

6.5 Limiting WIP based on people

Some teams might have a situation where people take work from start to finish, and it's not common or feasible to hand over the work to someone else. The workflow can still be a sequence of steps that the work goes through. In fact, this can be valuable to show what state the work is in right now, even though the same person is doing the work throughout the process.

A typical example of a team like this is in first-line support, in which you get a case from a customer in dire straits and you stay on it until it's solved. Such a case might go through the following phases: finding a workaround for the customer, filing a bug, testing a new release of the system with the bug fix, and finally closing the case. All these stages are probably handled by a single person on a support team.

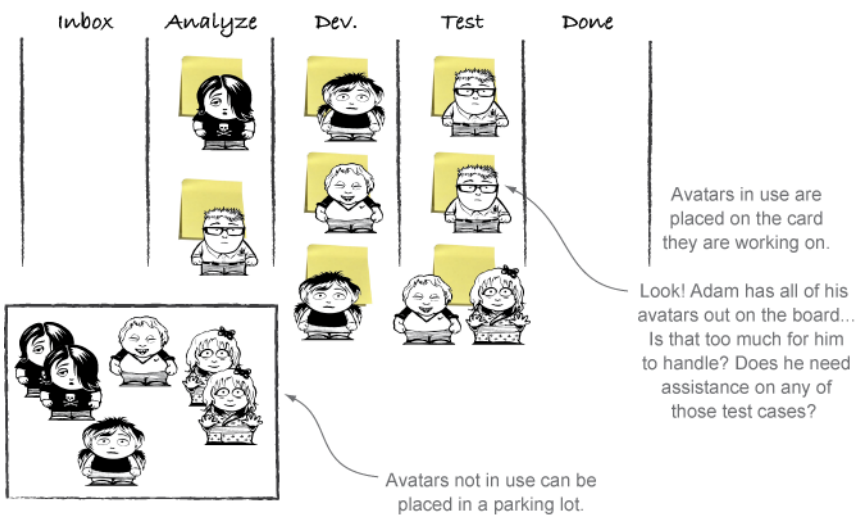
This calls for another strategy: some teams choose to limit the number of work items for each person. You focus not so much on optimizing for flow through the workflow, but rather on ensuring that no one takes on too much work and that everyone has things to do.

Another situation where limiting WIP per person is suitable is for fighting multitasking on all levels in the team. In this case, a WIP limit per person can help visualize that some people are involved in many work items,[3] which might be a driver behind the whole team having too much WIP. You can have conversations about how to prevent that, with a limit per person made explicit.

6.5.1 Common ways to limit WIP per person

We'll now take a closer look at some common ways to visualize people-based WIP limits.

THIS BOARD ISN'T BIG ENOUGH FOR ALL THESE AVATARS



[3] Such people are known as *hoarders* (see the next sidebar for a real-life example).

If you're using avatars to indicate who is working on what, a simple measure is to limit the number of avatars each person can have "in play" at the same time. You could, for example, print three avatars for each person, effectively setting the WIP limit per person to three. Now it's easy to see who's doing what and how many items each person is working with right now.

Poor Sean, he had too few avatars

On one team that Marcus coached, there was this guy: let's call him Sean. Sean had been involved with this application since before you were born. He practically built it himself. Every decision was in one way or another routed through him. You have probably met Sean a couple of times, or at least his type of person.

This slowed the team down considerably, because the team was quite big (eight people) and every work item had to be routed through Sean in order to be completed. When we created the kanban system, we printed only three avatars per team member to put a finger on this pain point.

And indeed, the first morning meeting took about 35 minutes, most of them spent waiting for Sean to place the three ("Only three? Can I have five more? Please?") stickies with what he should work on during the day.

After letting him sweat for a while, we had a discussion about the problems this situation caused for the team and the flow of items. To Sean's big surprise, it turned out that when he was on vacation or was sick, the team still got stuff out the door.

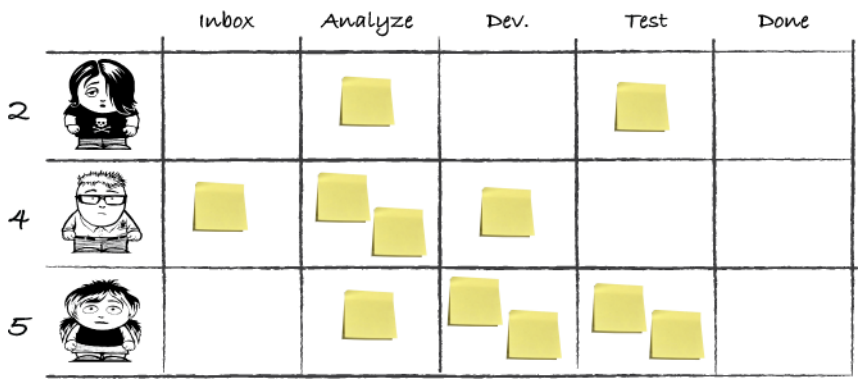
Maybe there were work items in which Sean didn't have to be involved? Maybe not having Sean involved in everything could increase the feeling of autonomy and mastery for other people on the team? Maybe that could free up Sean's time to do the complex stuff that he, in fact, was the only one on the team who knew?

It all ended well, but it was a bit shocking to poor Sean at first.

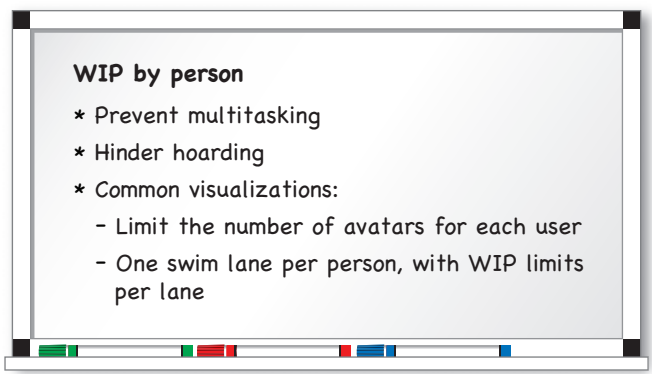
SWIM IN YOUR OWN LANE, PLEASE!

Another common way to limit WIP per person is to give each person a swim lane through the process. Then you can decide on a WIP limit for each swim lane, effectively allowing a certain number of items in any column of the board for that swim lane.

This approach can also be used to limit work per team on a multiteam board or for *teamlets* (parts of a team).



WARNING Remember that these strategies focus on making sure every person has enough to do; they don't help you much to get the work flowing to a finished state. You could start like this, but it's worth questioning the effectiveness of this approach to limiting WIP. After all, your customers seldom care if you're kept busy; they only want you to deliver stuff.



We have now given you a lot of principles, ideas, and practical tips for how to go about setting your WIP limit. The main point is that this is contextual and depends on your needs and situation. There are often a lot of questions around setting WIP limits, and we'll address some of the more common ones next.

6.6 Frequently asked questions

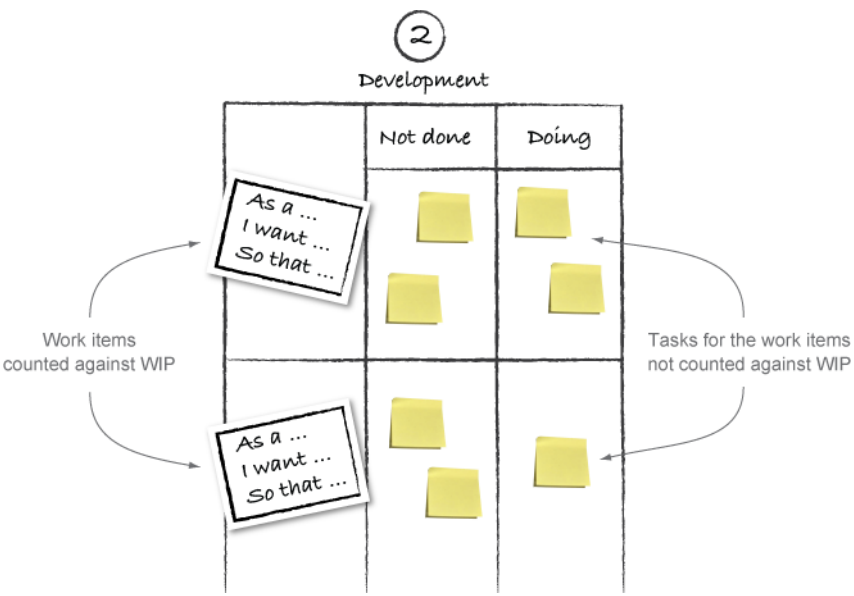
It's not strange that WIP limits trigger some questions, because the limit you select is something that depends on what *you* need and your situation. Sometimes, when WIP limits are described, they come across as something definitive; this adds to the confusion, because WIP limits should be moving targets.

6.6.1 Work items or tasks—what are you limiting?

Quite often, teams split up work items into tasks for certain steps in their workflow. For Development, it could be tasks like "implement data access," "write HTML page," and "complete business logic." In a Testing column, they could be tasks like "prepare test data," "write report," and "perform manual tests." Tasks are part of a work item and can be thought of as all the things you need to do to complete the work item in a given column.

Not until every task is completed in Development are the work items moved to the next step of the workflow. Some teams even divide the columns into subcolumns, forming a mini-board for the tasks.

Here you can see two work items going on (one in each row), and each work item has been split up into several tasks. The team counts the WIP limit against the work item, not the tasks that compose the work item.

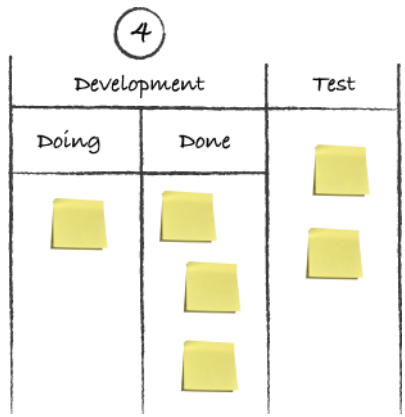


Most teams seem to work this way, counting their WIP limits against the work items. The tasks are only for tracking what you need to do in order to complete the work item and often aren't delivering customer value. It doesn't really matter how fast they flow across the board, if the customers still have to wait for anything useful to be delivered.

There could be times when, in certain columns, you might want to limit the number of tasks. For example, you may want to limit the level of multitasking or make sure you're always cooperating on certain tasks.

6.6.2 Should you count queues against the WIP limit?

Queues are often visualized as a column within another column. Ready for Test in the Test column is an example that we have used before.

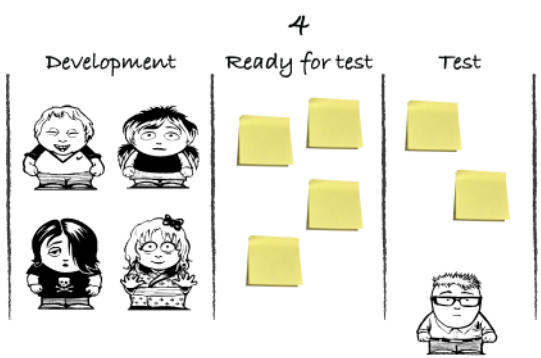


In the figure at right, you can see a WIP limit of 4 for the Development column. The column is then split into two sections: Doing and Done. The developers are working on one item and have three items completed, which keeps them at their WIP limit.

Another way of visualizing this queue could be to have a Ready to Test queue column in the Test column instead. That would then count against the WIP limit of the testing step.

How you visualize this on your team depends on what you want to achieve: that is, on who should "own" the tickets that wait to be tested. Does it help the team if the developers get the signal in their area, or is it more useful for the testers to count the items in the queue against their WIP limit? The choice is something that needs to be discussed with the team, but the practical function of the queue column is the same in either case: items stuck there will stall and hinder the flow, and people can't pull new work into columns before the queue. With this setup, you can easily see which items are done and which items the developers are still working on.

Finally, if you remember, back in section 6.4.1 we showed another way of doing this, where the queue column is a column by itself with its own WIP limit, as shown at right. In this case, the column has a WIP limit of its own, and it's not counted against anyone's limit. Maybe this is another technique that can help your team. The limit doesn't "steal" capacity from "your" column, but, again, the function of the queue is the same.



That way of visualizing the queue prevents the bottleneck that the single tester has become. A buffer is built before him to make sure he doesn't get flooded and yet always has work to do, which he can pick from the buffer (the testing queue) in front of him.

There are times when a queue should be counted against the WIP limit or even *is* the WIP limit itself (as in section 6.4.1 with the standalone queue column, for example). At other times, the distinction made between a queue and an *execution column* is a way to visualize detailed steps.

The answer to this section's question, "Should we count queues against the WIP limit?" is also, "It depends." For example, let's say you have noticed that testing is a bottleneck.[4] A queue might be a great way to ensure that the tester always has work that is ready to test. A WIP limit on the queue helps you make sure the queue doesn't get too long. With a WIP limit in place, you won't be able to flood the queue with work without breaking the WIP limit. When you see a queue filled to its WIP limit, it's time to discuss how to clear the blockage that is building up.

6.7 Exercise: WIP it, WIP it real good

The time has come for you to start discussing WIP limits. Again, this should be a team effort. Remember that there's no optimal WIP limit for your team; it's a tool that you can use to help you improve your work.

If you don't have an obvious WIP limit that you can come up with, first discuss the points in sections 6.3, 6.4, and 6.5. Here are some questions you can focus on:

- Should you have a single WIP limit for the complete board? Why would that be good for you?
- Can you start by limiting work in some columns? Which ones? Why?
- If you limit work per person, what kind of behavior would that drive or give you?
- Would swim lanes with WIP limits per lane help you in your situation?

If you don't know how to answer these questions, you should try something that seems appropriate, pay attention to what happens, and learn from that. When you come up with a good strategy and a number, start to discuss how this policy should be treated in your daily work:

- What kind of behavior are you encouraging by the way you've set up your limit?
- What should you do when you end up in a situation in which you're breaking the WIP limit?
- Should you count blocked/queued items against the limit?

As always, strive to start simple and make your approach more advanced as you see the need.

6.8 Summary

In this chapter we talked about finding and visualizing good WIP limits:

- There's no one right WIP limit for you and your team.
- Limiting WIP isn't the goal; improving flow is. WIP limits are merely a tool that helps find problems that hinder a better flow.

[4] Discovering and managing bottlenecks deserves a complete book on its own. But briefly, you might have a bottleneck if you see work accumulated before the bottleneck and if steps after the bottleneck are starved for work.

- A lower WIP limit will move your work faster and also bring problems to your attention more quickly. You want a low WIP limit, but not too low.
- When setting out to find a WIP limit, start simple—maybe as simple as the slogan "Stop starting, start finishing."
- You can limit the WIP for the whole team/workflow:
  - This approach can help you improve collaboration.
  - It helps you keep focus on finishing work items before taking on new ones.
- You can limit WIP based on column:
  - This puts focus on flowing items through the workflow.
  - It can help to manage bottlenecks.
  - Maybe you already have a hunch about what you need to improve and can start limiting WIP for that column.
  - Work can also be limited by estimated size (story points, for example).
- You can limit WIP based on people:
  - Limiting the number of avatars that can be in play is one way to easily accomplish this.
  - Using swim lanes is another.
  - This can help you prevent too much multitasking and people being involved in every item on the board.
- Most teams have a WIP limit on the work-item level, not the task level.

WIP limits are ways to improve the flow of your workflow, but what is that? It's one of the main concepts of Lean that we haven't yet talked about. And it's the topic of the next chapter.

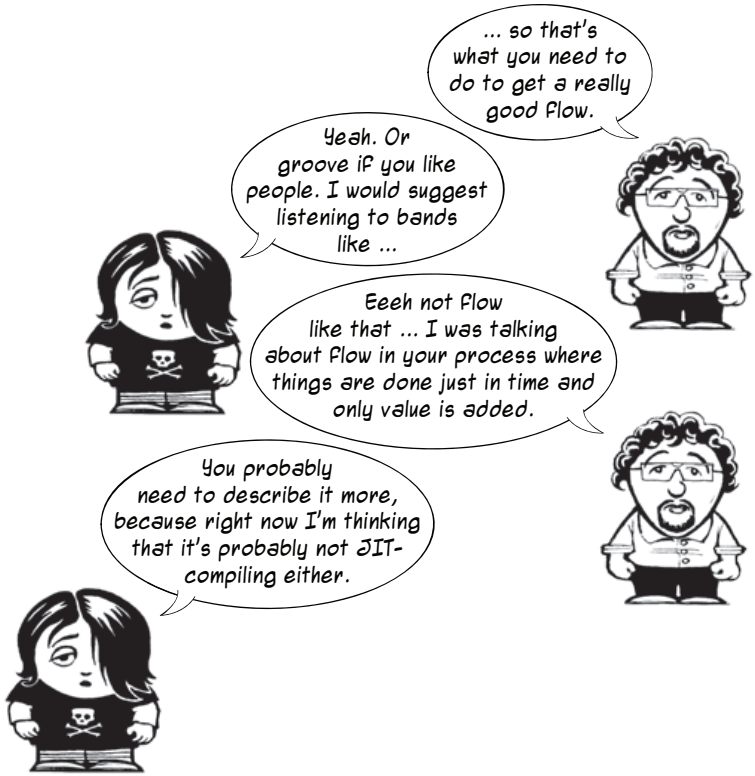
7 Managing flow

This chapter covers

- What continuous flow is and why you want it
- Eliminating waste to achieve flow
- Managing flow with kanban
- Using daily standups to help the work flow
- Choosing what to work on next
- Managing bottlenecks using the Theory of Constraints

Flow, or rather *one-piece continuous flow*, has been the cornerstone in Toyota's[1] production vision for decades. A one-piece continuous flow is a system in which each part of work that creates value for the customer moves from one value-adding step in the process directly to the next, and so on until it reaches the customer, without any waiting time or batching between those steps. Instead of building up inventories *just in case*, things are produced only when needed, *just in time*—at the right place and in the right quantity: no more, no less.

[1] Toyota is the company that pioneered the Toyota Production System (TPS), which Lean is based on.



This continuous flow turns every process into a tightly linked chain in which everything is connected. There is nowhere for a problem to hide, no inventory of other work or products to work on if something stops or breaks; so when things break, you immediately know what's happened, and you're forced to solve the problem together. It forces people to think, and through this they become better team members and people.

In this chapter, we'll take a closer look at the benefits of the flow approach to organizing your work. You'll learn about waste elimination and cycle time, but also how to actually manage the flow by keeping the work moving and by using models such as the Theory of Constraints to improve your process. We'll take a look at some common practices that can help you focus and improve the flow in your process, such as reducing waiting time, removing blockers, and using cross-functional teams. There are some practices that many teams use that can further help you focus on flow, such as daily standups and policies around what to work on next; we'll check into those as well.

But let's start at the beginning and see why flow is something that's worth striving for and focusing your attention on.

7.1 Why flow?

Toyota is by no means the only, or even the first, company to chase the continuous-flow ideal. People and companies have been striving toward this vision for centuries. And why wouldn't they? If one-piece continuous flow is achieved, you have no delays, no queues, no batches, no inventory, no waiting. You have no over-production; you only deliver what is actually requested and needed by the customer, and you deliver it immediately when it's needed.

You've seen in previous chapters that this gives you flexibility and responsiveness, better risk management, faster feedback leading to improved quality (building the right thing and building it right), increased predictability leading to increased trust, fewer expedite requests, and faster continuous delivery of value.

As if that isn't enough, flow also exposes improvement opportunities and provides you with a vision for your improvement efforts: faster flow.

What can you do to get a faster flow, then? Let's look at a strategy that has helped Toyota for a long time: eliminating waste.

7.1.1 Eliminating waste

Toyota has spent more time perfecting, and has come closer to the vision of, one-piece continuous flow than any other company. An important part of that journey has been a focus on eliminating waste.

> *All we are doing is looking at the time line from the moment the customer gives us an order to the point when we collect the cash. And we are reducing that time line by removing the non-value-added wastes.*
>
> —Taiichi Ohno[2]

Another way of expressing this is to ask, "What's stopping the work from flowing?" Applying this perspective, you examine your processes from the customer's point of view. What does the customer want from this process? The customer can be the end customer or the internal customer in the next phase of production. What the customer wants has value. Everything else is waste. Some steps in a process add value, others don't. This is true for all processes.

[2] Taiichi Ohno, *Toyota Production System: Beyond Large-Scale Production* (Productivity Press, 1988, http://amzn.com/0915299143), p. ix.

Early on, Toyota identified seven categories of waste that applied to manufacturing; and later the company applied the same thinking to other areas, such as product development. Different industries have since then made similar categories of waste for their particular contexts. Mary and Tom Poppendieck did this for software development through their seminal works *Lean Software Development: An Agile Toolkit* (Addison-Wesley Professional, 2003) and *Implementing Lean Software Development: From Concept to Cash* (Addison-Wesley Professional, 2006).

7.1.2 The seven wastes of software development

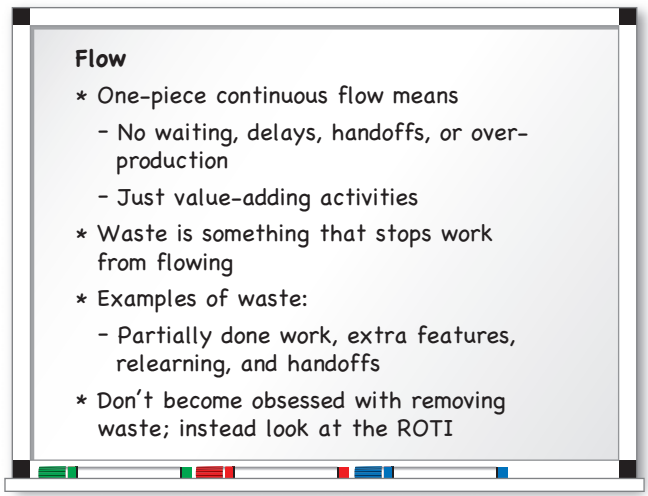
Let's take a closer look at the seven different kinds of wastes that Mary and Tom Poppendieck have identified:

- *Partially done work*—Work that isn't completely done is in reality just work in process. It lies around waiting for you to complete it and adds to your WIP. See chapter 5 for more on this.
- *Extra features*—Taiichi Ohno famously said, "There's no greater waste than overproduction." A lot of the software being produced in the world isn't really used or valued by the customer. According to the famous CHAOS report performed by The Standish Group in the early 2000s, 45% of the features of software applications were never used. This suggests that a lot of waste could be avoided with better understanding of what the customer needs: that is, building the right thing. On an individual level, if you've ever thought "This may come in handy" or "Maybe they want this to be configurable," you know what we mean.
- *Relearning*—Software development is, to a great extent, learning. Failing to remember mistakes you've made before and having to redo them (and relearn the solution) is a great waste. This can happen both in teams and for individuals.
- *Handoffs*—When you hand work from one person to another, a lot of extra work is created in order to convey necessary information to the next person. Even with that extra work, a lot of information will be lost in the handoff.
- *Delays*—Delays create extra work, remember? No? Go read section 5.2.2, "Delay causes extra work," without delay.
- *Task switching*—We talked earlier about context switching (see chapter 5) and the wasteful effect it can have on your focus and capacity.
- *Defects*—Defects are work that comes back to you because you did something wrong the first time around. Not only does this create extra work, but typically that kind of work comes at a bad time, slowing down the things you're working on right now. See section 5.2.5, "Lower quality."

A WORD FROM THE COACH Some people in the kanban community think there has been an unhealthy obsession with waste within Lean software development and that waste is a red herring. According to them, you should instead be focusing on reducing time to delivery and risk management; waste will be identified and removed as a part of this process. We're not sure we see the

conflict, but you should of course avoid unhealthy obsessions and heavily faith-based approaches in general. Part of the confusion comes when people look to see if certain actions are waste.

For example, is a daily standup meeting to coordinate team activities adding value, or is it waste? If it's a value-adding activity, why don't you do more of it? Why not just meet all day long? The answer is that the activity in itself isn't a value-add or waste; it's the return of time invested (ROTI) to help you improve your understanding or deliver customer value that determines if it's waste or not.



Eliminating waste and limiting WIP are different ways of achieving flow. But how do you apply these principles in practice to help the work to flow?

7.2 Helping the work to flow

One of the core practices of kanban is to manage the flow of work through each state in the workflow. This means monitoring and measuring how the work flows, but also keeping the work moving. You can take several different approaches to help the work to flow.

7.2.1 Limiting work in process

Limiting WIP is of course an essential practice when it comes to keeping the work moving. The fewer things in process, the faster the work will flow and the more improvement opportunities you'll discover. You might remember Little's law, which states that the more WIP you have, the longer the lead time for each item is. You can read much more about this in chapter 5.

Flow vs. resource utilization

Limiting your WIP gives you better flow. But is that what you want? Optimizing for flow is a strategy decision. That decision is a trade-off; to achieve better flow, you might end up with people sitting idle from time to time. Is that acceptable in order to get better flow through your process?

There are types of work and situations for which the opposite is true: you strive to use your *resources* optimally instead. One example is a mill for aluminum. That kind of equipment operates at extreme heat and takes a long time to heat up. It's expensive to turn off the mill. You want the melting furnace to be running all the time, and you want to have a lot of material ready for the furnace to handle (lots of WIP). You might even want it running when no customers are waiting for the aluminum. You're optimizing for resource utilization.

In the other case, optimizing for flow, an example is the fire department. They have a lot of slack (waiting, preparing, and training) in order to be ready to go immediately and put out a fire when it happens. We, as a society, accept that and are paying firefighters to sit idle, because we don't want them busy when we call and ask them to put out a fire. To optimize for flow, we're accepting the fact that they're not busy all the time. That kind of behavior is typically found in high-risk environments.

Optimizing for flow over resource utilization is a strategy decision, but by focusing on flow efficiency you can reduce a lot of superfluous work and waste and actually improve resource efficiency too. In the book *This Is Lean: Resolving the Efficiency Paradox* (Rheologica, 2012), researchers Niklas Modig and Pär Åhlström define *Lean* as an operations strategy that aims to increase resource efficiency through focusing on increasing flow efficiency.

7.2.2 Reducing waiting time

Do you have work items sitting in the board's Done columns, waiting to be pulled to the next state? Are items sitting in states like Waiting to Deploy or Todo? It isn't unusual to see work spending more time waiting to be worked on than actually being worked on. How can this waiting time be reduced?

If you don't already have an explicit way of showing that work is done in a particular state and ready for the next, this should be your first step. This makes the waiting visible and signals to others that the work item is ready to be pulled. It also makes it easier to track and measure the amount of time spent in waiting. Collecting this data and analyzing it may give you insight into what can be improved. If you have a hard time convincing others that time is wasted in waiting, measuring it like this can serve as an eye opener.

Do you start the work from scratch, or is it started or prepared somewhere else (what we typically call *upstream* from the kanban board)? Maybe the work is sitting there, waiting for you to start it. Are you the last people to touch the work before it