

The  
Pragmatic  
Programmers

# Real-World Kanban

## Second Edition

Do Less, Accomplish More  
with Lean Thinking



**Mattias Skarin**

Foreword by Henrik Kniberg

*edited by Michael Swaine*



## What Readers Said About the First Edition of *Real-World Kanban*

This is a very practical and to-the-point book on how to implement the Kanban method. The case studies turn theory into practice in a very practical and to-the-point way. This is a must-read for managers interested in improving end-to-end product development.

► **Håkan Forss**

Agile Coach, King

*Real-World Kanban* is a great collection of case studies plus a practical summary of Lean principles for software development. It shows how adjusting development to focus on flow and feedback greatly improves efficiency by increasing the value—rather than the quantity—of the output. The book is loaded with examples of well-conceived visualization that provides the situational awareness vital for success in fast-moving environments.

► **Mary Poppendieck**

Poppendieck, LLC

If you want to know what Kanban looks like in practice, this book is for you!

► **Arne Rook**

Kanban Pioneer “Dr. Rock” @ Jimdo



We've left this page blank to make the page numbers the same in the electronic and paper books.

We tried just leaving it out, but then people wrote us to ask about the missing pages.

Anyway, Eddy the Gerbil wanted to say "hello."



# Real-World Kanban, Second Edition

Do Less, Accomplish More with Lean Thinking

Mattias Skarin

The Pragmatic Bookshelf

Dallas, Texas



See our complete catalog of hands-on, practical,  
and Pragmatic content for software developers:

<https://pragprog.com>

Sales, volume licensing, and support:

[support@pragprog.com](mailto:support@pragprog.com)

Derivative works, AI training and testing,  
international translations, and other rights:

[rights@pragprog.com](mailto:rights@pragprog.com)

The team that produced this book includes:

Publisher: Dave Thomas

COO: Janet Furlow

Executive Editor: Susannah Davidson

Development Editor: Michael Swaine

Copy Editor: L. Sakhi MacMillan

Indexing: Potomac Indexing, LLC

Layout: Gilson Graphics

Copyright © 2025 The Pragmatic Programmers, LLC.

All rights reserved. No part of this publication may be reproduced by any means, nor may any derivative works be made from this publication, nor may this content be used to train or test an artificial intelligence system, without the prior consent of the publisher.

When we are aware that a term used in this book is claimed as a trademark, the designation is printed with an initial capital letter or in all capitals.

The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg, and the linking g device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions or for damages that may result from the use of information (including program listings) contained herein.

ISBN-13: 979-8-88865-159-9

Encoded using recycled binary digits.

Book version: P1.0—July 2025

# Contents

	<b>Foreword</b> . . . . .	<b>ix</b>
	<b>Acknowledgments</b> . . . . .	<b>xi</b>
	<b>Take Charge and Make Changes</b> . . . . .	<b>xiii</b>
<b>1.</b>	<b>Leading with Kanban</b> . . . . .	<b>1</b>
	Putting Your Flow Glasses On	2
	Improving Flow with Kanban	3
	Applying Kanban in Knowledge Work	5
	Next Steps	9
<b>2.</b>	<b>Using Kanban in the Enterprise</b> . . . . .	<b>11</b>
	The Challenge: Improving Time to Market	11
	How We Got Started	13
	How the Process Worked	16
	What Lessons We Learned	29
	Comparing Now and Before	40
	Make Your Own Improvements	42
	Next Steps	42
<b>3.</b>	<b>Using Kanban to Manage Change</b> . . . . .	<b>43</b>
	The Challenge: Managing Dependencies Without Burning Out	43
	How We Got Started	44
	How Our Process Worked	45
	How We Continuously Improved	51
	What Lessons We Learned	53

	Comparing Now and Before	53
	Make Your Own Improvements	54
	Next Steps	55
<b>4.</b>	<b>Using Kanban to Save a Derailing Project</b>	<b>57</b>
	The Challenge: Restoring Trust by Solving the Right Problem	57
	How We Got Started	59
	How Our Process Worked	64
	How We Continuously Improved	67
	What Lessons We Learned	69
	Comparing Now and Before	71
	Make Your Own Improvements	72
	Next Steps	72
<b>5.</b>	<b>Using Kanban in the Back Office: Outside IT</b>	<b>75</b>
	The Challenge: Keeping Up with Growth	75
	How We Got Started	76
	How Our Process Worked	77
	How We Continuously Improved	81
	What Lessons We Learned	83
	Comparing Now and Before	84
	Make Your Own Improvements	85
	Next Steps	85
<b>6.</b>	<b>Using Kanban in Capital Markets</b>	<b>87</b>
	The Challenge: Solving the Seniority Catch-22	87
	Don't Wait for the Perfect Timing	89
	A Detour to Alternative Process Frameworks	91
	Going Back to Kanban	92
	Lessons Learned	93
	Make Your Own Improvements	93
	Next Steps	93
<b>7.</b>	<b>Using Kanban in Remote Work</b>	<b>95</b>
	Building Remote-First Kanban Teams That Work Well	95
	For Your Journey Ahead	98

<b>A1. Kanban Board Examples</b>	<b>99</b>
Product Development	99
Software Development Flow	100
Development Team with Multiple Clients	101
Development Team Practicing Prediction	102
Full Value Chain	103
Mixing Product Discovery and Delivery	104
Progress by Architecture	105
Business Operations	106
Corporate Legal	107
Product Roadmap for Complex System (HW+SW)	108
System Administration Team	109
DevOps for Online Platform	110
Release Management	111
First-Line Support	112
Marketing and Sales—Sales Team from Lead to Deal	113
Sales Team Respond to RFP	114
Unpacking Vision and High-Level Goals	115
<b>Bibliography</b>	<b>117</b>
<b>Index</b>	<b>119</b>

# Foreword

---

Kanban is a bit like the Chinese board game Go—a few moments to learn, a lifetime to master. The rules of Go are really simple, yet there are hundreds of books on how to play the game well. And even if you somehow read every one of those books, you'll still be a lousy player if you don't practice!

However, there is a nice way to cheat—to learn from other people's experiences! Watch game replays. Listen to grandmasters analyzing their moves and telling the story of the game as it unfolds. Of course, you still need to practice. But studying experts in action will give you a huge boost and help you avoid the most common mistakes.

And that's what this book is—a series of expert-level game replays for Kanban implementations.

When it comes to Kanban, Mattias is the real deal! A true practitioner, with years of experience knee-deep in the trenches helping organizations improve. I particularly remember our first coaching engagement together, where the introduction of cross-functional feature teams and limiting work in progress enabled a company to repeatedly build new products in three to four months instead of two years. That was such an interesting case that Mary and Tom Poppendieck included the story in their book [\*Leading Lean Software Development\* \[PP09\]](#).

Readers of my books know I like real-life examples, with warts and all. And that's exactly what *Real-World Kanban* is! The core of this book is five short stories about real-life Kanban implementations—the context, the challenges, what they did, and what they learned. It is full of photos and drawings, nuggets of wisdom and practical advice, and a sprinkling of theory. Mattias does a great job illustrating the mechanics of Kanban, such as how to organize the boards and use hard data like cycle time and cumulate flow diagrams, while also emphasizing the importance of soft factors such as motivation, communication, and leadership culture.

Five stories (instead of one) means less depth per story, but in return you see patterns! And that's the unique value of this book—seeing how the same overall pattern of thinking was applied in five different contexts.

Storytelling is the most ancient and effective way of conveying knowledge, and that shines clear in this book. Enjoy!

**Henrik Kniberg**

Agile/Lean coach and author of *Lean from the Trenches*

# Acknowledgments

---

This book would not have come about without the contribution of the people in the case studies. Thank you to Håkan Forss, Mike Burrows, Henrik Kniberg, and Tanuj Shroff for their efforts during technical review. Also, I owe my thanks to Fahmida, Mike, and Susannah from Pragmatic for crafting the chapters. A final thanks to Judith for her help in polishing the English wording.



# Take Charge and Make Changes

---

Agile undoubtedly revolutionized software development, but something got lost: encouragement for organizations to self-improve. For many organizations, agile meant using frameworks, Scrum, SAFe, and similar tools. But unfortunately, after the initial excitement of the shiny new toy, improvements often stalled. My hypothesis is that organizations need to learn to self-improve, but instead they simply learned to follow a script.

This book is for you who want to challenge the status quo, for you whose aim is set at achieving continuous improvement.

If you would like to do agile better, this book is for you. If you're looking for ideas on how to improve efficiency without adding resources, this is for you. If you'd like to learn from real case studies and pick up a few tips and tricks for your own improvement, this is for you. If you're wondering how to use agile in a non-software domain, this is for you.

This book offers five case studies on how real-world companies used Kanban and lean thinking to improve time to market, product quality, and cross-department collaboration and teamwork.

As I write this second edition, nine years after the first, I will admit that one reason it took such a long time was that I was convinced that the developments in product development methodology would have overtaken the book in two or three years, making the first edition of the book more of an anecdotal time capsule of "the old ways."

Rereading the book today, with the perspective of product development in 2025, I can see that it challenges most organizations' comfort zones.

A few use cases where the book has relevance:

- How to improve end-to-end value delivery by 2x
- How to improve beyond scaling agile in IT

- How to, in our context, find a constructive way out of a death march project
- How to drive change critical to shared success in the organization around you

I've always been curious to find out how a team that started their improvement journey using Kanban would fare a few years down the road. I'm thrilled to be able to share a story of such a team.

Finally, this second edition includes a rich set of Kanban board examples as a bonus, which should be helpful for those of you who would like to set up a Kanban system.

## Who Should Read This Book?

This book doesn't have any how-to recipes. This is a case-study book—each chapter shows how we found the right solution for each problem. This book was written with managers and business leaders in mind. The case studies provide helpful ideas for managers who run agile teams and agile teams who want closer interactions with people in business units and other operations outside IT. Business leaders who want transparency in what goes on in IT and managers interested in learning how we ran a multi-team development project without a formal project office should check out this book.

Scrum masters will pick up guerilla tips and tricks for solving problems with other teams, and senior managers will learn how to coach leaders to collaborate over the value chain.

## Helpful Books

Here's a list of the many excellent and in-depth books out there that I'd recommend for further reading:

### Books on Kanban

- Anderson, David J. *Kanban* [And10]
- Burrows, Mike. *Kanban from the Inside* [Bur14]
- Hammarberg, Marcus, and Joakim Sunden. *Kanban in Action* [HS14]
- Kniberg, Henrik, and Mattias Skarin. *Kanban and Scrum: Making the Most of Both* [KS09]

### Books on Lean

- Kniberg, Henrik. *Lean from the trenches* [Kni11]

- Liker, Jeffrey. *The Toyota Way* [Lik04]
- Modig, Niclas. *This is Lean* [MA12]
- Poppendieck, Mary, and Tom Poppendieck. *Lean Software Development* [PP03]
- Reinertsen, Donald. *The Principles of Product Development Flow* [Rei09]

## Online Resources

This book also has its own web page.<sup>1</sup> Check it out—you'll find the book forum, where you can talk with other readers and with me. If you find any mistakes, please report them on the errata page.

---

1. <https://pragprog.com/titles/mskanban2/real-world-kanban-second-edition/>

# Leading with Kanban

---

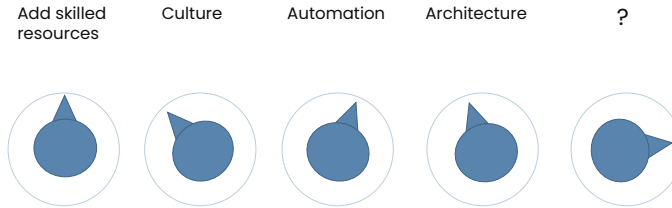
You have just been given your new leadership position. You're full of enthusiasm about the new journey you intend to take the organization on. You knew things weren't optimal when you signed up (after all, that's why you're there). But the more you learn about how things really work, the more worried you become. Things are less mature than you thought across the board, from the technology to the organization. But you're determined to make a difference. The question is, where do you start?

Words from an old management book came to mind, "Get the right people on the bus, and then think of where to drive it to." It sounded profound when you first read it, but in the current situation it's not really helpful. Sure, you'll probably be able to convince a few talented people to come on board, but that will take time. Time is costly; there's a time window to act in. Not only is it a matter of what the people who hired you expect from you, it's also a matter of trust from the people you lead—they're watching. Doing nothing is not an option. You realize you'll have to work with the people you have. And you'll need to make your improvement choices wisely.

You're determined to improve the system. What kind of system levers or "knobs" do you have at your disposal? You make a small napkin sketch, as shown in the [figure on page 2](#).

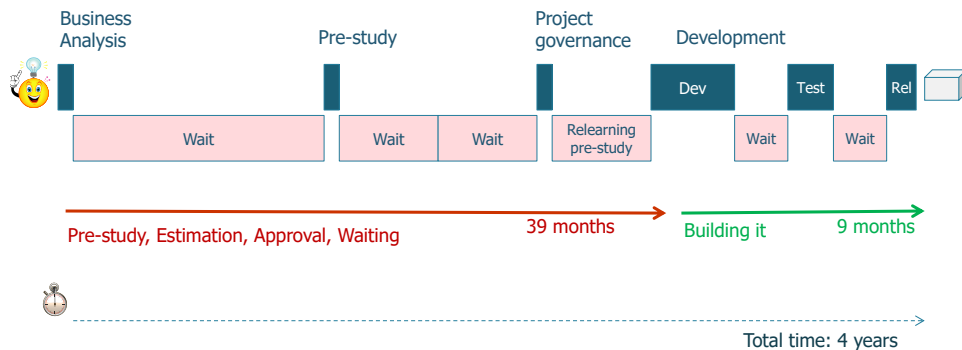
When you review the different knobs, two things dawn on you: 1) improvements of importance are inside each and every one of them and 2) the time-lag factor. It can take years before you see any tangible effect. Unfortunately, the organization isn't really known for its patience, so even if the knobs on your sketch would help, it simply would take too long.

Is there any knob you've left out? You draw a final knob with a question mark over it. What could this be?



## Putting Your Flow Glasses On

You sift through old trainings and recall a few words from colleagues in other industries. Someone was really enthusiastic about lean and flow. Others were over the top about agile. At the time, you filed those ideas under “interesting, but I’m an expert in product development, which is different.” In light of your situation, where you need to make wise choices on what to improve instead of following the process created by someone else, you do what you do best—you take on the problem. You pull out some data from the product development IT systems, and you draw the journey of an idea—from start to in production. This is what you see:

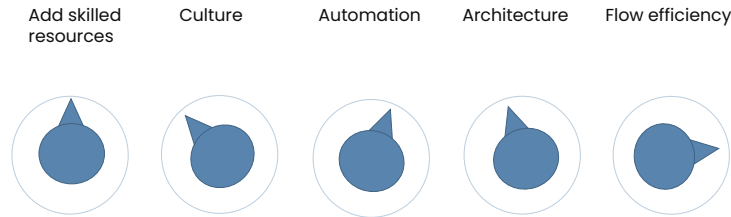


As you eyeball the data, it hits you. As an organization, you’re not effective. From the customer’s perspective, you’re slow. And yet, everyone is working round-the-clock. What’s going on? You realize you’re on to something.

You toy with the idea, “What if we could reduce, maybe even eliminate the waiting time (in pink), then we could theoretically be as fast as the sum of the green sections—with the same people, and within the resources we have.” You’ve found the key! You realized that the variable overlooked in almost

every organization or process change you've been exposed to was the net effect on time—the effect of the change on lead time was always overlooked.

The new perspective through which you are now seeing operations is what lean coaches call *flow*. You've found the quicker, nimbler system lever *flow efficiency*.



The challenge now becomes How can I turn flow efficiency into practice? Here's where Kanban comes in. The beauty of Kanban is the simplicity. It's easy to start, easy to operate, and most importantly, Kanban can be run inside any operational domain. You don't need to change your process or organization to get started; you start where you are. Making good use of Kanban is easy. Kanban is of course only a tool; it needs to be run by people who want to improve. But in the right hands, the hands of people doing the work, spiced up with knowledge of flow, Kanban can be a very powerful improvement tool indeed.

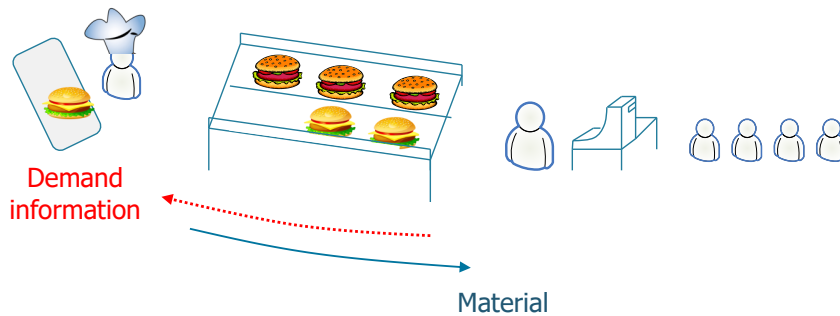
## Improving Flow with Kanban

Kanban was originally a tool used by Toyota to balance demand and capacity across the value chain. The idea is simple: a Kanban card is sent upstream when parts are needed. It's only then that production for the needed number of parts is done. The arrival of a new card is a signal to produce more parts, and a lack of cards is a signal to stop. The number of cards is limited to prevent overproduction and to reduce the parts needed in production. Keeping half-finished parts ties up valuable working capital that can otherwise be used for investments.

Let's take a look at how it works in the familiar case of a burger joint.

The challenge our burger joint faces is keeping fresh burgers ready to serve while the number of customers varies over the day. A free spot on the tray is a signal to our chef to cook more burgers. When the tray is full, our chef can

do other tasks—maybe prepare for the upcoming week, clean up, or help out elsewhere.



The number of cooked burgers on the tray is a buffer balancing our chef's ability to cook burgers with our ability to handle sudden surges in demand. Using this simple mechanism, we don't cook more burgers than we need, and we can adapt easily to changing demand. (In real life, the batch size of how many burgers to produce varies throughout the day to adapt to high peaks, such as lunchtime.)

#### The Kanban Rules

The Kanban rules (as applied in manufacturing) are in fact much more interesting than the Kanban cards:

1. A later process tells an earlier process when new items are required.
2. The earlier process produces what the later process needs.
3. No items can be made or moved without a Kanban.
4. Defects are not passed on to the next stage.
5. The number of Kanbans is reduced carefully to lower inventories and to reveal problems.



These Kanban rules tell you a lot about the intent behind Kanban cards as used in manufacturing. By understanding the behaviors they drive, we can learn how to apply Kanban wisely in a different setting, such as in product development.

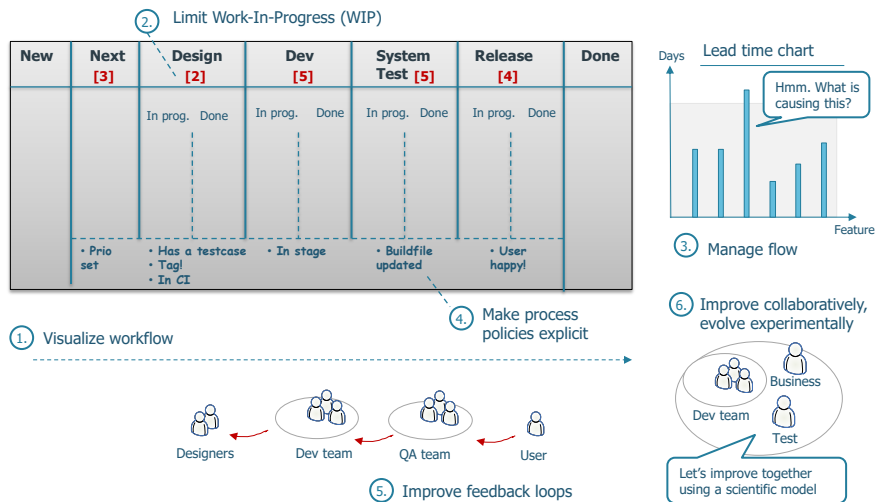
It's dead simple. It's so simple that no one needs to think about the process. It's a natural part of work. It's easy to see at a glance if you need to keep up

or slow down. During times of stress, there's less risk of misunderstandings and errors.

Another advantage is flexibility. A production line with Kanbans is simple to change. Any change can be made locally. Just reassemble the stations, change the number of Kanbans, and you're done.

## Applying Kanban in Knowledge Work

Now that we have a bit of historical perspective on how Kanban was originally used in manufacturing, let's see how the six core practices of Kanban apply to knowledge work:



1. *Visualize workflow.* Knowledge work is largely invisible, often hidden in hard drives and in email inboxes. Visualizing workflow allows us as a team to act and learn based on a shared overview. This is helpful to spot bottlenecks and recurring quality problems.

2. *Limit work-in-progress (WIP).* The purpose is to balance demand and capability. By limiting the work-in-progress, we allow our teams to work at a sustainable pace with quality output. Limiting WIP is often the first step to shift the emphasis from starting to finishing.

3. *Manage flow.* To improve, we manage our constraints and measure flow. The two most common measurements for flow are throughput and lead time.

4. *Make process policies explicit.* It's hard to make improvements if every team member has a different standard. An explicit policy is necessary so that there's



a shared agreement among team members working with the Kanban board. An example is a definition of “done” per column before moving work forward.

5. *Implement feedback loops.* A Kanban system will only reflect your side of the story—how you see your quality. You’ll need to implement a feedback loop as well to help you learn if you’re getting it right during product development.

6. *Improve collaboratively, evolve experimentally.* Evolve using problem solving, experiments, and scientific methods. The big idea behind the Kanban method as applied to knowledge work is to improve evolutionarily from the current state using small steps.

So what lessons from Kanban in manufacturing can we apply to product development? First is the value of time. Time erodes away value. A product idea might be great now but less so in a few years’ time.

The second component that’s even more valuable in product development is transparency. What we work on in product development is often hidden on hard drives, in email inboxes, in people’s minds. Transparency helps teams act as one; it also helps in spotting areas for improvements. From a change agent’s perspective, you no longer need to “sell the problem.” The problem is already apparent! The conversation can directly be shifted to “what can we do about it?”

## Choose Pull over Push

*Faced with the choice of changing one’s mind and proving there is no need to do so, almost everyone gets busy on the proof.* —John Kenneth Galbraith

While Kanban shows what to improve, it’s our job to do something about it. It takes the will to improve, as well as courage and skill. Without the will to do better, improvement stalls. Without courage, we won’t dare to experiment, challenge old thinking, and try new paths.

One of the essential leadership skills needed is change leadership. An organization’s experience of a leader can be seen as the product of the leader’s ability to single out the right problems to solve and how he went about solving them. The latter is a good indication of the leader’s preferred change leadership style. Attacking the right problem with a flawed change leadership style won’t fly.

The rule of thumb for a successful change leadership is choosing *pull* over *push*. This can be challenging for a leader who is used to calling the shots and getting things done. Now he needs to sell the problem and find a path to

get his teams to address it. The clue to *pull* is understanding the problem and “what’s in it for me?” Note that choosing *pull* when it comes to change leadership is not the same as giving up when you meet resistance—it merely means trying a new path.

The best piece of change leadership advice I got from a senior leader was this: *For new teams you meet, don’t push change; instead, help them solve a problem.*

One of the main contributions from agile is the art of chunking down a hard-to-solve problem into bite-size pieces. Those small improvements add up. Over time, the net effect can be achieving the seemingly impossible. The opposite is often always true—big changes most often fail.<sup>1</sup> So master the art of making small improvements, often. And to be clear, there are times you do need to make system-level changes at a higher level. Have the courage to do so! The difference is that when you do, you can point at what you tried to do yourself first. And that makes a ton of difference to the engagement you’ll get at higher levels.

All teams in this book followed this tactic.

## Choosing a Transformation Pattern that Beats Random

I’ll use the word *transformation* to describe a larger change an enterprise might engage in. It can be lean/agile transformation, scaled agile transformation, or <insert favorite framework here> transformation. Now, I really want to stress that agile and lean do offer substantial benefits for product development organizations, something I hope you see in the case studies chapters. What often fails in a transformation is generally not the tool itself but, rather, how it’s wielded.

What most often fails is the change tactics used. Unfortunately, it’s the combined experience that persists in organizational memory. When you look back at its history, an organization doesn’t often distinguish between the tool, the change tactic, or its own lack of accountability in the process.

Let me describe four different change tactics used for transformations, all of which I have observed at various clients. My description of them might be a bit provocative, and yes, I’m oversimplifying reality to make it easier for you to recognize patterns.

Now if you are a change agent running a transformation and feel provoked, then lay down the book, have a laugh, and call me silly. It’s okay! And for those of you who want to skip the problem statement and jump directly to

---

1. <https://hbr.org/2023/11/why-big-projects-fail-and-how-to-give-yours-a-better-chance-of-success>

field-tested leadership practices that give transformation capabilities to organizations, check out the Practice Library at [activeagileleadership.com](https://activeagileleadership.com).<sup>2</sup>

My intent in sharing these different transformation tactics is to inspire change agents to try out a different change path than the one they would normally take, or better still, consider challenging the instructions given by clients. If this nudges some of you to try out a different transformation change tactic, that's great!

Here are four typical transformation change patterns:

1. Fitting the organization to a model (and hoping for the best)	<p>The model can be any process framework (SAFe, LeSS, Scrum, XP, or &lt;insert your favorite&gt;). Let's just assume for the moment that they're all tools and carry some value.</p> <p>The question of importance is this: at the end of the day, after you applied it, did you improve anything? Or did you make things <i>worse</i>?</p> <p>This change pattern is attractive because it's easy to get started with. Just apply the process. The trouble with this change pattern is that it doesn't consider the starting position of the organization, nor does it consider improving beyond the basics of the process framework. The change agent's job is simplified to teach the model.</p>
2. Random walk of changes driven by coaches (with little or no product development experience)	<p>This is recognizable by too much emphasis on individual coaching and empowerment and not enough emphasis on structure in any form. Sometimes, structure is even portrayed as the enemy of creativity.</p> <p>You can recognize this change pattern by the lack of strategy and accountability. The change process is opaque; transparency is nonexistent.</p> <p>While each and every change might make sense in isolation, in combination they are hard to grasp and lack strategy.</p>

---

2. <https://activeagileleadership.com/practicelibrary>

3. Fitting the organization to extracted pieces of a model, driven by context	<p>This is recognized by extracting patterns from a process framework based on the organization's areas of improvements and then scaling those out.</p> <p>This pattern produces smaller changes, which is a good thing. The challenge is to bridge the gap between model and result—to demonstrate that each change yields positive results before scaling.</p>
4. Flow-grounded improvements	<p>Improvements are focused on areas that improve flow. This makes each improvement a more precise bet.</p> <p>(An extra level of maturity can follow if your organization has already mastered end-to-end flow efficiency. The next step is value efficiency. Like panning for gold, it's learning to rapidly test product ideas and finding the ones that customers love.)</p>

I'll now go further and offer an axiom:

If you follow the preceding transformation patterns 1 or 2, the best you can hope for is random success.

For transformational change agents that value repeat success, patterns 3 or 4 yield substantially better odds. While there's always a risk a change won't work out, the level of precision you'll get on your bets is higher, and with that, you'll gain a few positive experiences to start a positive spiral.

If you wonder if I myself have failed to drive change, the answer is yes! I share the preceding patterns as some of my hard-earned lessons. What you do with them is up to you!

Risk is inherent in change. The reverse is also true: if you aren't taking any risk, you aren't driving any change.

## Next Steps

Okay, that's it for now with a bit of background theory.

Let's walk through our case studies and see what really happened.

# Using Kanban in the Enterprise

---

It's easy to lose focus on what is important in software development—to make useful products and make it simple to do so. Let's take a look at how Enterprise Kanban helped a traditional company do just that.

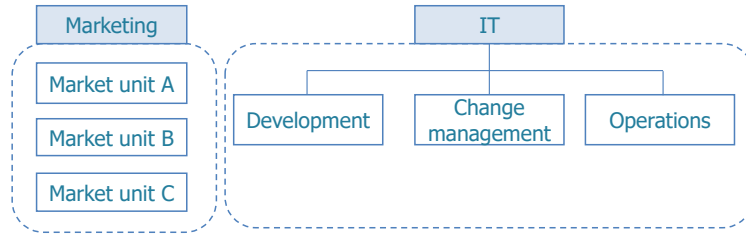
Very few companies start off improvements with a clean slate. They carry legacies—people, technology, roles, culture, market share, and processes. The legacy proves the company was successful at some point, but it now results in heavy processes and structures, which slow down productivity and make things hard to change. Bureaucracy has increased. At the same time, hidden quality problems make it tempting for managers to call for even more control and coordination.

Let's see how a company learned to define product ideas, keep track of status, and release new features faster with Enterprise Kanban.

## The Challenge: Improving Time to Market

We look at Company H, which has been in business for 100 years, for our first case study. One of the early pioneers in computing, the company has a hefty tech stack—more than 300 systems—dating from the 1970s. This is a challenge because Company H is competing with newer companies that aren't carrying the same kind of technological and organizational baggage. The competitors are fast-moving and aggressive, and Company H has to become more effective and modernize its products to compete.

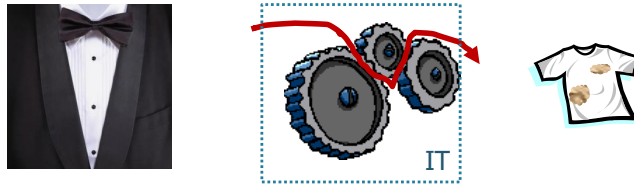
As you can see in the diagram [shown on page 12](#), the marketing department at Company H has three units, each targeting a specific market segment. IT is also split into three units: development, change management, and operations.



Company H employs a total of 650 people, with roughly 100 involved in new product development. All the employees were somehow, directly or indirectly, affected by the Kanban board.

What was Company H's key challenge? To ship modern products that appeal to buyers faster to remain competitive in the marketplace. One problem was communication and handovers.

"I requested a suit, but all I got was a lousy T-shirt," a marketing manager said, regarding the company's communication problems.



The original product concept was getting diluted or lost somewhere along the way. Product managers weren't clearly communicating to the developers the things that made the product unique, or the developers were under pressure to ship quickly and were cutting corners. The developers felt as if they were merely small cogs in the machinery and were missing the big picture of what they were creating.

### Why Change Was Necessary

Internally, many people within Company H felt improvements had stalled. The development teams had adopted agile to initial success, but big projects still dragged on for too long. Company H had just aborted an earlier project aimed at renewing the product platform, which had been overdue for well over 18 months and was still far from finished. Just because the teams are agile doesn't mean the company is.

No one seemed to know the exact state of the product ideas under development. These ideas existed partially in several Scrum teams' product backlogs. In some cases, they were in different stages of testing at the same time. Adding an Enterprise Kanban board to see the true progress of new product ideas was a natural step. This would drastically simplify marketing and would allow us to see what stage the product idea was in.

We wanted the team to take more pride in and responsibility for the overall results—what we call *product idea success*. Let's look at how we did something about it using the Kanban board.

## How We Got Started

After clarifying and agreeing on why change was needed, it was time to take the first steps. It's always tricky figuring out how to get started. We approached this challenge by selling our ideas to the people involved. Obviously, one of the ideas was to start using Kanban to visualize end-to-end flow.

We invited the people and the teams involved and presented our findings and reasons and the four or five changes we wanted to try out. We then asked for feedback to see which ones they might consider trying out. We did this by presenting the ideas one by one and asking people to *thumb vote* on their readiness to try out the idea.

Thumb voting is a very simple decision-making technique and helps teams jump into action:

- *Thumb up* means "Yes! Let's go!"
- *Thumb to the side* means "Hmm, I'm unsure but willing to give it a try."
- *Thumb down* means "No way, this is crazy!"

Using this technique, four out of our five proposed changes received a go-ahead. Asking for feedback on the changes before deploying them might seem strange. What if we had gotten a No! on all of our ideas? Let's consider the worst case: we move forward, only to find silent resistance from the people we're depending on. By presenting the reasoning behind the proposed change (the *why*) and talking about what we'll do, we treat people as intelligent, thinking beings. We get better solutions and consider unforeseen constraints before we even get started.

Decision made, we set up our Enterprise Kanban board to help us visualize the process flow. We needed to see how the teams moved from a simple idea to delivering something valuable to the customer.

### It's Important to Ask for Feedback Before a Change



As a general rule, people are more likely to agree to an idea or change if they get to weigh in first. People are more willing to accept an alternative plan if they feel their concerns have been heard as part of the process. If you get a negative response, then ask for suggestions. Doing nothing is never the better option. Letting other ideas bubble up always is.



### Set Up the Kanban Board

We had to first decide what to put on the board. Each department had its own nomenclature and preferred level of detail to describe various tasks. We decided to include only product ideas and features on the Kanban board. Product ideas represented a unit of something with a sales value. Features represented a change in the product. By getting marketing and IT to agree on what to put on the board, we now had a shared language to communicate status. If someone from marketing wanted to know the status of a particular idea, the information was on the Kanban board.



---

### Shared Language Across Departments Is Important

---



To address problems early, we need a common vocabulary to define the granularity of the features we're working with and a way to communicate the feature's current status. If each function has its own nomenclature and level of granularity, too much information will be lost in translation. This should be the first thing you address if you have multiple groups involved.

---

We filled the board with ongoing and upcoming product ideas. Mapping current sprint items to ongoing product ideas was a fun challenge, but it took a day or two to get right. Getting the overview was difficult because each team's sprint backlog contained different parts of the product under development.

At this point, we already saw the Kanban board's benefits. We spotted items from product backlogs that wouldn't enhance the overall product or actually benefit the teams. The board also made it easier to see the number of product ideas floating around as well as the real development status of individual ideas.

Where to put our Kanban board was another important decision. The board tied together four functions: marketing, development, change management, and operations. It made sense to put the board in a corridor outside the development teams' location. That way, the team members interacting with the board most often were the closest to it, and people who didn't interact directly with the board saw it at least twice a week during the stand-up meetings. (We cover more about this in [Focus Behavior with the Board, on page 24.](#))

Most of the people involved with the various concepts passed by the board at least once a week. But we were lucky that most of the people worked in the same location. If we had been spread out geographically, we would have needed an electronic version of the board or replicated physical boards in each location to make sure we all saw the same picture.

### Focus on Flow, Not Sprints

With the board in place, we asked the development teams to shift gears. Instead of constantly running sprints, we asked the developers to think about *continuous flow*. This would reduce wait time but also allow developers to work on product ideas until the ideas were finished and of the correct quality as opposed to shipping them just because the sprint had finished. It represented a shift from being *date-driven* to being *quality-driven*.

At first, the development teams were cautious about this change. In their view, sprints *worked*. But they were also keen to be involved earlier in the product development process and feel less like a cog in the machinery, so they agreed to give flow a try.

### Reintroducing Sprint Planning

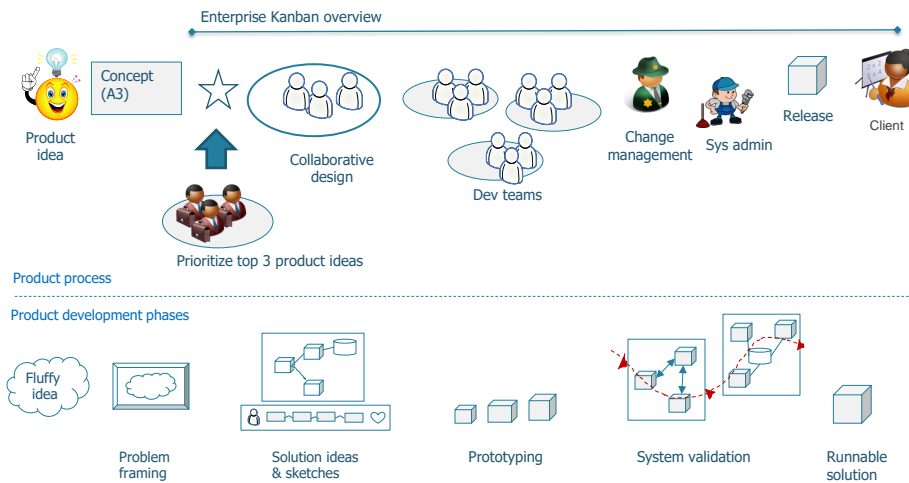


After a couple of months, some of the development teams reintroduced light sprint planning because they needed a way to see what each team member was working on. They also wanted a way to work together when slicing features into stories. This was okay and fit our overall goal of continuous flow because the teams didn't spend sprint planning trying to estimate how much work they could fit into sprints. Instead, they used sprint planning to focus on product ideas.

Getting the board set up and putting the ideas on it was just the beginning. Let's look at how we defined the workflow and how each step corresponded to the board.

## How the Process Worked

For a process to be useful, the people who use it have to take ownership. To do that, the process has to be simple. The following figure shows an overview of the Enterprise Kanban workflow:



Let's take a closer look at some of the elements—namely, *concept* and *collaborative design*.

## A Quick Explanation About Concepts

A concept refers to a product idea written down on an A3 (12"x16") piece of paper. With concepts, it's easier to share the big-picture idea across multiple teams and to abort ideas early in the development process if no one cares about them. Concepts can also help to decentralize risk because teams can make tradeoff decisions without having to get permission from someone else. Concepts help us maintain the integrity of the original idea as it passes through each phase of development.

Traditionally, product managers or product owners are responsible for product decisions. We took a slightly different approach. We wanted the most passionate person behind the idea to drive it, regardless of role. But this came with a condition: *"You want it, you make it happen. No one will make the product happen for you—there's nothing to hand over to anyone. We think you are the right person to handle it because you're passionate about it."*

The concept is just an idea at this point. The details come out of the collaborative design meeting.

## Adopt Collaborative Design

We set up collaborative design to achieve three things. The first goal was to reduce wait time. By having multiple minds looking at a problem, we would get multiple perspectives quickly—no need to wait for the next iteration to find out whether something was doable.

The second goal was to eliminate the "we're only a small cog in the machinery" feeling among team members. Since one member of each team participated, that person would bring back to the team an understanding of the problem addressed plus the reasoning behind design decisions.

The third goal was to get *creative height* during design. We wanted to avoid turning a breakdown into a lame exercise in fitting the product idea into the existing architecture. Our goal was to always deliver two solutions to any problem.

A facilitator—generally a Scrum master from one of the development teams, who had received specific coaching—calls and runs the collaborative design meeting. One developer from each team participates in the meeting, with specialists getting pulled in as necessary. A specialist may be necessary if the design requires integrating with outside teams, for example.



During the meeting the concept owner paints the picture, or describes the idea and walks through the concept. The meeting participants carve out two to four potential solutions in the *discover* phase. Then the group digs into each solution as part of the *explore* phase. Finally, the group discusses the pros and cons of each solution and selects one or two options to move forward with. The facilitator is responsible for moving the group through each step, balancing coming up with new ideas and focusing on details.

A good facilitator ensures that multiple options are explored, especially for ideas that were rejected unintentionally or that got lost during the conversation. Facilitators should also inspire team members to think about the problem from different angles: “This is a valid solution, but what would the simplest solution look like?” It’s also important to transcribe the discussion. Ideas are often discarded unintentionally and end up getting lost. Transcribing lets participants review earlier conversations and solutions.

Facilitators also pay attention to participants and pick personalities that balance each other. Few ideas emerge from a homogeneous group. The goal is to expose different angles and encourage participants to build on each other’s ideas. One way to balance the team is to have a rotating membership.

## Work with the Kanban Board

Now that we’re familiar with the overall workflow, let’s see how it translates to the Enterprise Kanban board. As we saw earlier, a product owner writes down a product idea, a concept. This concept is then prioritized by the head of each marketing department before it’s placed on the Kanban board.

### Don't Come with Predecided Solutions!



In a few cases, we let a senior developer and the concept owner pair up and walk through the idea together before coming to the collaborative design session. That didn't work out well at all, because it seemed like a solution was already predecided by the time of the meeting. "Why am I here to give you ideas? It seems like you've already decided on the solution," a developer said during one such meeting.

So we learned to bring in fresh problem statements rather than half-finished solutions.

A concept represents a marketable idea cut across multiple systems. To give a sense of scale, it could be sized in months. Each concept in this situation included a description of the multiple features needed to deliver the intended market impact. Each feature was split into stories tracked by individual development teams. In total, we had three granularities of work:



Concepts – marketable idea, tracked on Kanban board.



Features – value for at least one user. Part of a concept.



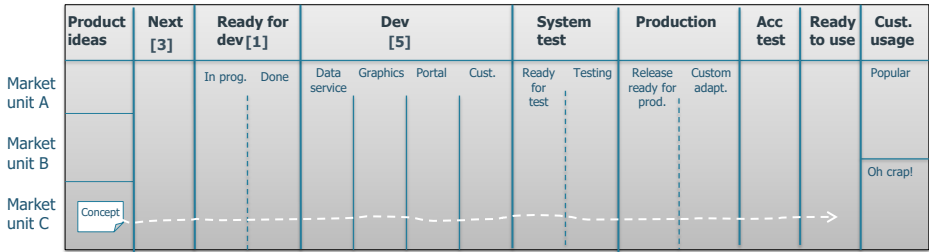
Story – deliverable slice on a feature in a development team, tracked by each team.

We decided to track concepts across the board because they represented the marketable product ideas as our customers saw them. And we wanted these concepts to be the focal point of conversations between marketing and IT. At first, we also experimented with keeping each feature in a concept across the board. But then we thought better of it because we felt that we lost the overview of what was important. The features could easily be looked up anyway, since all the concepts were up on the wall right next to the Kanban board.

Take a look at the board [shown on page 20](#).

As mentioned earlier, what flowed across the board was concepts, each with a designated concept owner.

How did we make product portfolio changes? Each marketing department was responsible for a defined customer segment and maintained a product portfolio of both features in production and ideas under development. If they



discovered that the portfolio needed to be improved, the department either wrote a new concept or asked a concept owner to make the necessary changes to an existing concept to improve overall experience.

Let's take a closer look at each column on our Kanban board.

*Product Ideas*

Each marketing department was responsible for keeping two product ideas prepared here. For a product idea to exist on the board, it had to have a prepared concept.

*Next*

These were the next three product ideas the team would work on. This is where we began lead-time measurements. From this point on, marketing could not insert new concepts or make major changes to scope. Marketing *could* cancel the idea, in which case it would be moved to the Oh Crap! section at the end of the board. At Company H, managers from each marketing department and the head of development met in front of the board every 14 days to review the priorities. This was an opportunity to discuss and agree on whether an investment in technical debt made sense for the teams.

*Ready for Dev*

At this point we evaluated different solution options and how they fit the problem and decided which teams would be impacted. As we saw in [Adopt Collaborative Design, on page 17](#), at least one representative from each team participated in the meeting.

*Dev*

The product ideas progressed over multiple teams. Before a product idea moved to the System Test column, the teams and the concept owner validated the product's usability and fit for purpose.

### *System Test*

This was basic verification to see whether the product worked from a system perspective. Integration and deployment on production such as platform, maintainability, data, and stability over time would be verified.

### *Production*

The release was moved into production. Customer-specific branding and configurations would be added if necessary.

### *Acceptance Test*

Validation of the final experience by the concept owner.

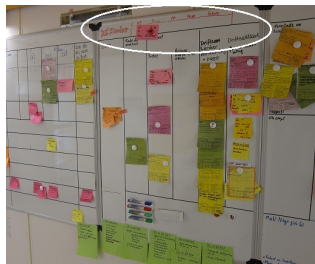
### *Ready to Use*

Ready to be taken into use by customer.

### *Customer Usage*

This represented feedback from customers. If they liked it and used it, it was deemed Popular; if it didn't work out, it was deemed Oh Crap!

We learned early on that some key impediments could not be tied directly to a single product idea because they spanned several ideas. To ensure that we acted on them, we added a section on top of the board, as shown in the following image, where each development team could signal if some factor was impeding their progress.



While it may seem trivial, this sent an important signal that we did care about blockers and we weren't going to proceed until we had fixed them. It was an important leadership signal showing that these things mattered, and we frequently used it early in our Kanban implementation. As we solved a couple of the key blocking issues, we used it less frequently. Currently, it's rarely used. It's still on the board mainly to assure the teams that if they raise a serious concern, they'll be heard. This section is an unfiltered communication channel all the way to both the head of marketing and the head of development.

You might wonder why some of these blockers weren't addressed before. We'd been using Scrum and development teams for some time. A simple explanation

is that each of these problems was too big for a single team to solve. All required cooperation across teams and sometimes functions to address. Now we were able to focus across functions to address them.

## How We Decided What to Invest In

We now had several product ideas—concepts, if you like—on the board. Each marketing department had prioritized its requests, but the team had yet to decide which concepts would get promoted from the Product Ideas column into the Next column.

One of the most commonly used methods for investment decisions is to calculate *return on investment* (ROI) for new development projects. Traditionally, costs are estimated by forecasting the number of *man-hours* needed to complete the project. ROI helps you decide whether the project will be a profitable investment and figure out a sensible IT budget for it.

We invariably had to make tradeoff decisions on what to develop due to budget and resource constraints. So we would do a value versus effort judgment one way or the other. The problem happened when our effort was largely directed to reducing cost uncertainty rather than value. The value of the product idea normally carries higher uncertainty than the cost. So spending too much effort estimating the cost side of the equation isn't effort well spent, because you're addressing the wrong uncertainty.

Rather than trying to estimate effort and cost this early in development, we emphasized reducing product value uncertainty instead.

Our approach to estimating the cost component was simple. We used two simple assumptions: first, cost, of which headcount is a main component that tends to remain fairly stable over time. Changes do happen, but they're usually rare. Second, *time through the system = effort consumed*. We used estimated lead time for the product idea to learn how much effort it had consumed.

With the cost range covered, focus could now be shifted to the value component of each product idea to make the decision about what to invest in.

One key insight that came to light during this process was that it was important to ensure that each marketing department got its fair share of the development capacity. But this didn't need to be resolved through budgeting. (See [Make Sure Each Marketing Department Got Its Fair Share, on page 23.](#))

This approach made it possible to avoid paralysis by analysis and to work with low overheads.



### Match Estimate Effort with the Decision



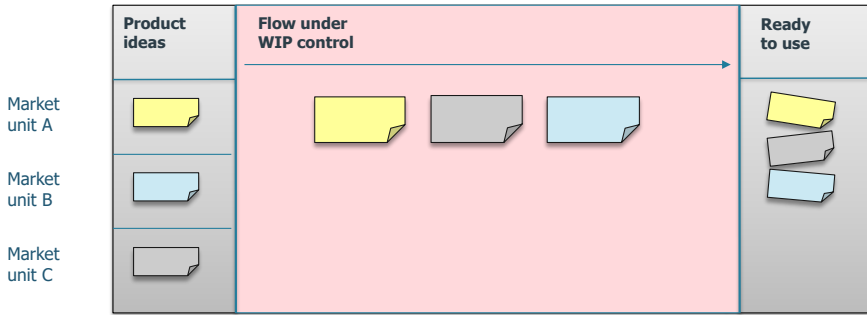
An all-too-common mistake I see product development departments make is determining cost and time estimates on too fine a granularity. Nothing's wrong with making ROI calls, but the time invested in making them should be linked to the decision you're trying to make.

Clarify the decision you're trying to make first; then decide what makes a reasonable investment to make your decision.

### Make Sure Each Marketing Department Got Its Fair Share

Each marketing department focused on identifying the next product idea that would most likely succeed in the market. And the department heads met in front of the Kanban board every 14 days to review priorities. This guaranteed transparency.

Each marketing department contributed an equal amount ( $\frac{1}{3}$  each) toward the IT budget. This meant that each department got its proportional part of development and was guaranteed to get every third product idea. This rule could be adjusted if the heads of each department agreed that a certain product idea or improvement would be more valuable to the company. For example, all the departments in Company H overruled the “every third product idea” rule when they agreed that a concept that would improve performance was more important than other concepts on the board.



### How Teams Decided What to Work On

A question that surfaced early was what we should do about other work that was not directly related to the Kanban board. To avoid micromanagement

and overloading the board, we decided on a decision rule to help organize the board:

- Fifty percent of work should be product-idea oriented (taken from the Kanban board).
- Twenty percent would be improvements (if there were none from the enterprise board—this was left to the team to decide).
- Twenty percent would be bug fixing.
- Ten percent would be quick fixes, answering questions, and so on.

Each team was given the mandate to make the call on whether to pick up work when approached by external parties as long as they kept these rough guidelines. We also clarified that we wanted each team to keep visualizing the work they did on their team board. This allowed both teams and stakeholders to spot deviations to the rule. One example of what we learned was that one team was working on very big features, each of which seemed to drag on forever. We broke down these features into smaller slices. This allowed the team to make progress easier and provided greater flexibility to absorb unexpected changes, such as helping out other teams.

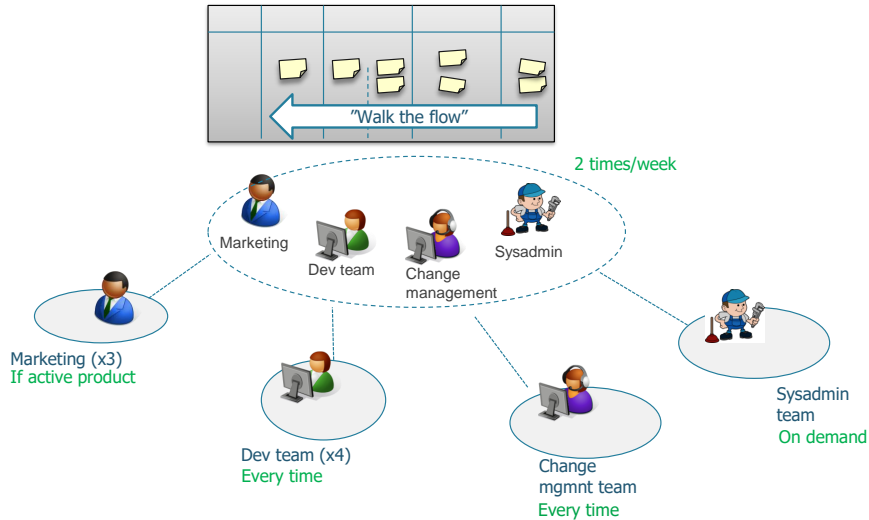
## Focus Behavior with the Board

To create a shared understanding of current status and to address problems, we created a stand-up meeting. We decided on 15 minutes twice a week. This ensured that stakeholders working in other parts of the building would get a regular overview of the ideas at work. It also ensured that key stakeholders knew they could get in touch with each other at regular intervals (see the figure [shown on page 25](#)).

At first, we asked for a (minimum) representation by each function at the stand-up. That meant three people from marketing (one per function), six from development (one per team plus the head of engineering), one from change management, and one from operations. Specialists were pulled in on an as-needed basis. Our facilitator at these meetings and the owner of the Enterprise Kanban board was our head of product development.

Operations remarked after a while that there was rarely an item on the board that required their input, so we simplified overall attendance to “If you have something of interest on the board, you come to the stand-up.” And we found that worked better.

Under the new rule, concept owners (marketing people who had active product ideas under development) would always be at the meeting, as well as one



representative from each development team (if they were working on an active concept). We pulled in specialized resources, such as system administrators, or outside teams as needed.

To ensure the stand-up meetings ran smoothly, we stuck to a simple agenda. First, we walked the flow. The goal of this step was to find out whether any blockers were preventing progress. We walked this from the back of the board forward. If a blocker was identified, we asked who would address the situation. One person would be assigned the responsibility for dealing with each blocker. We avoided discussing the problem during the stand-up meeting because that should be done afterward. Before ending the meeting, we recapped important points (for example, who owned each blocking event).

We officially ended meetings with a “Thank you, everyone—we’re done for today.”

#### Keeping Meetings Short



With anywhere from 10 to 20 people from different teams and divisions in front of the board, keeping these meetings short was key. We learned to arrive prepared and to study the board in advance, instead of tripping over the typical “oh, what was this about again?” queries while people patiently waited. It worked—we’re still holding stand-up meetings twice a week.

The idea of having a stand-up is to see and act on problems. This means the forum needs to have decision-making authority on both product-level calls (Should we release this now or later? Should Product A stand back for Product B?) and technical issues (Who are the right people to address this technical problem?). Make sure the relevant people are at the meeting, or grant decision-making authority to people attending. This way, the stand-up truly becomes a decision-making forum and not just a vehicle for status reporting.

### How We Continuously Improved

Take a look at the workflow again in [How the Process Worked, on page 16](#). The workflow takes the product idea from concept to collaborative design, development, change management, and product release. Under Enterprise Kanban, however, *you run with the idea all the way to working with the client*. This means we're not finished when the product is released. We're done when the customer uses the product.

The most valuable things learned in product development come from user feedback regarding the product. Whether the feedback is good or bad, hiding it delays the learning process (at best) and can be detrimental to product development (at worst). It's easy to get distracted when you measure parts rather than the whole, thus failing to see the forest for the trees. If you want to improve at the system level, you need to measure and share feedback from this level, too.

Continuous improvement can be run in many ways. We decided that the most important information came from the usage (or in the worst-case scenario, non-usage) of our products. So we made this information the foundation of our improvements. We started by visualizing the outcome of our development at the end of our Kanban board (see the following figure).

	Product ideas	Next [3]	Ready for dev [1]		Dev [5]				System test		Production		Acc test	Ready to use	Cust. usage
Market unit A			In prog.	Done	Data service	Graphics	Portal	Cust.	Ready for test	Testing	Release ready for prod.	Custom adapt.			Popular
Market unit B															Oh crap!
Market unit C															

This became our most important feedback loop—were we delivering things of value? If the customer didn't like the product idea, it was put into the Oh Crap! area. Conversely, if the customer liked the product idea and it became

frequently used, it would go into the Popular area. If an Oh Crap! event occurred, we brought together the concept owner, the teams involved, and the facilitator to perform a root cause analysis. This then became the input to the changes we needed to make.

One of the key changes we made when we started Enterprise Kanban was to replace the sprint demo with a company demo, which we ran one day before release. Releases ran at four-week intervals. At this event, new product ideas were demoed by the teams, allowing people throughout Company H to see what was going out.

But we added a step to the demo agenda. Based on the previous release, Marketing would demo how products were being used and share customer feedback and comments. This was much appreciated by the development teams and gave engineers the opportunity to learn about how the products were being used by the clients.

When the teams used Scrum, they ran sprint demos. Unfortunately, these demos inevitably took on a development bias at the expense of user value. When we shifted to company demos to focus on working product ideas, the conversations became more about market reaction and product usage. Sprint demos were replaced by continuous feedback on product fit between the concept owner and the active development team.

---

#### Continuous Feedback

---



If for any reason you're not able to continuously validate the usefulness of a product under development, alarm bells should ring. Regardless of your choice of development method, this is your first clue that something will go wrong. Every new product needs at least one revision before getting it right—this is your key to corrective action. The art of slicing (turning big things into small) and continuous validation of working software are two key components that can help you get it right.

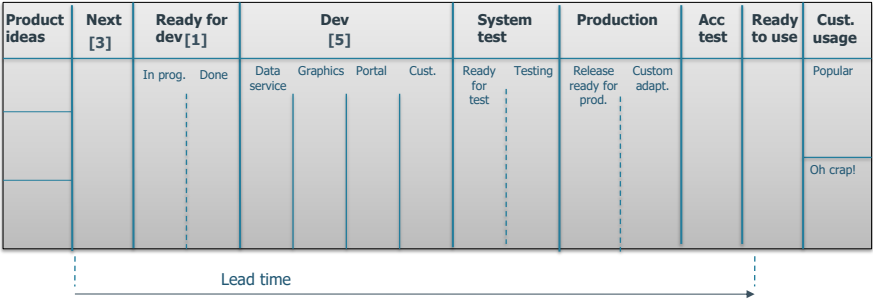
---

### Use Metrics Effectively

In software, early indicators of success or failure often show up in the form of observations by people close to the problem. Experience helps in detecting important signals from noise. It's easy to be biased, however, since it's often hard to gauge how many improvements have been made. Measurements are not about measuring perfection; they're about having fact-based feedback on whether you're improving.

We tracked two metrics: lead time (including components) and percentage of concepts that reached the Popular stage versus the Oh Crap! stage. If we improved time to market, we would get feedback on flow. The metrics would also show how the teams were doing on delivering value. These measurements would give us feedback on flow (if we improved time to market) and how we were doing on delivering value.

Lead-time measurement starts when the product idea enters WIP (the Next column for us) and stops when the customer can use it (the Ready to Use column), as shown in the figure that follows. The reason for selecting these boundaries is because they are under our control. So extraneous actions such as how quickly the client accessed and used the product are less likely to cause noise in our data.



Once a month, the manager of the IT department (our Kanban board owner) pulled together one representative from each development team for an *improvement pulse*, a quick retrospective, in front of the Kanban board. The agenda for these was very simple. The manager would review metrics (lead time and customer usage feedback), review the board (is it clear, easy to view, and useful?), and make necessary changes to the board. The meetings let managers take the pulse of the team and were typically very quick—about 15 minutes—with changes to the board made immediately.

The improvement pulse may change things such as the templates being used for the Kanban cards, refine lead-time metrics, or even insert/remove/reinsert swim lanes (the horizontal rows) on the board.

Team issues or major blockers were rarely discussed during the monthly improvement pulse. Why? Because they had already been addressed. If a team or a product idea got blocked for some reason (such as performance issues or release issues), the issues would have already received attention

and would have been addressed. Thus, we rarely had to spend time discussing fixes for blocking issues at our monthly improvement pulse.

## Act and Share Information

We complemented the metrics we collected with several visual indicators, such as blocking events, queues, age of product ideas, and estimates of where the team put the most effort. These indicators helped management and teams discuss and take action while the information and the corresponding chain of events were still fresh in people's minds. Trying to resolve the problem a month later would have been a challenge because it would've been much harder to recall the chain of events.

We estimated how much effort each team spent and presented that information in a small section at the top of the Kanban board. You can see the area of the board highlighted in the following image.



Each team updated and reviewed this section in the presence of the IT managers during the monthly retrospective. As shown in the following figure, each team changed the size of the columns for each category to show their effort allocation. If there was a big discrepancy between the team's estimates and the target figure, the IT manager could ask about possible causes and determine what potential actions should be taken.

The visualization was a remarkably simple mechanism that let us track extraordinary events as well as situations where teams were pushed in the wrong direction (for political or personal reasons). This mechanism replaced time reporting as a tool to learn where the team spent time.

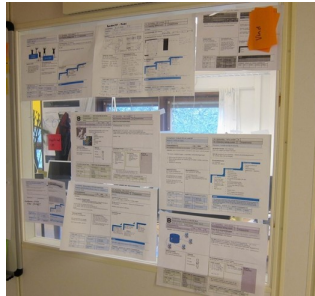
## What Lessons We Learned

We learned a number of lessons during this process. We improved inflow quality, found our time to market, located our first improvement opportunity,

stopped doing late changes, and improved lead time by a factor of 2. Let's look at each lesson in detail.

## Improve Product Ideas

In the beginning, we kept prospective new product ideas on a wall next to the Kanban board. Because each product idea was presented as an A3 card, we pinned the promising prospects next to the Kanban board (see the following figure).



The first time I reviewed these prospective new product ideas, I noticed that 40 percent of them didn't have answers to the key questions, which would be essential to have a meaningful conversation with the development team. For example, impact was often missing. As we discussed earlier, concept owners must be prepared to discuss the details with developers before the concept owners can start working on the idea.

To fix this, the software team leads were given the authority to request that concept owners supply the missing information. Once we did this, we noticed a small but important behavior change—for the first time, the teams saw that they could ask for quality input, much in the same way concept owners expected product features to work the first time they tested them. This helped emphasize that we were serious about the quality-first mindset.

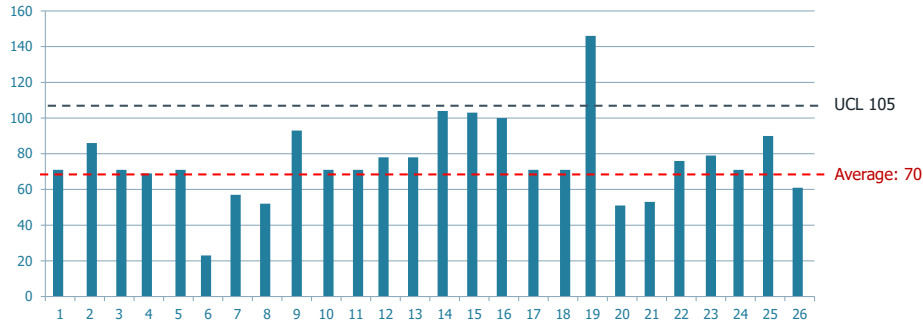
While it would be unreasonable to expect that every product idea would be prepared to perfection this early in the learning process, you can't help but wonder what would have happened if we had tried to develop and release those product ideas.

## How We Found Our Time to Market

To answer the age-old question of "When can I get my stuff?" we sampled the lead time for released products and figured out where the 95th percentile was. This is often referred to as the *upper control limit* (UCL).



The following figure shows our first sampling of the delivery times for new product ideas. Each bar represents a delivered product idea, and the vertical scale shows the number of days it took for the product idea to become a shipped product (lead time). A UCL of 105 means 95 out of 100 product ideas would get delivered within 105 days.



This helped marketing manage client expectations and know when to begin preparations for a new product if they wanted to hit a certain time frame or season.

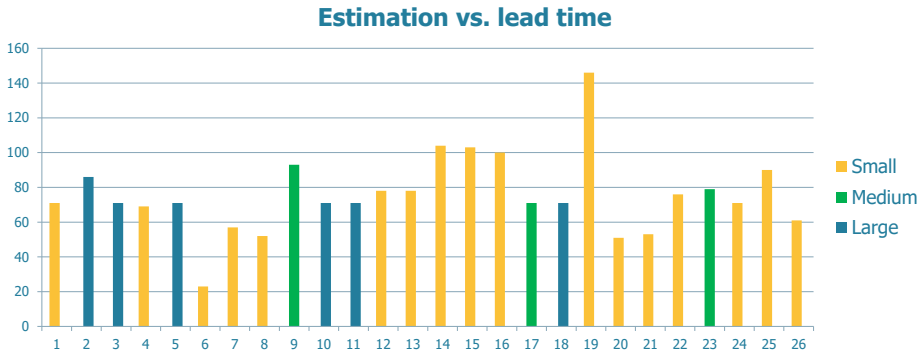
I frequently hear the argument for needing up-front estimates because some items are bigger than others. That's very true—some items are indeed bigger. If you look at the chart again, you'll see that some bars are taller than others, meaning they took a longer time to deliver. The interesting question here is how the actual delivery times correlate to developers' up-front estimates.

To help answer this, we had developers estimate sizes using the following buckets: small (two to three days), medium (one to three weeks), and large (longer than one month). We then correlated the initial sizing estimates with lead-time output.

Take a look at the [figure on page 32](#). Is the initial sizing a good predictor of when you can expect to get your stuff?

In our case, the surprising truth was a resounding “no!” Judging by the data in the chart, we can argue that there was only one estimation bucket (or two, depending on whether you see the longest data point as a random event).

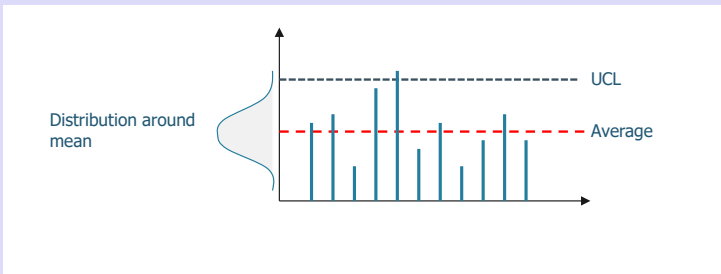
If size wasn't the dominant factor in lead time, what was? In our case there were two factors: the wait time for release and the wait time to get access to specialized skills.



We found out later, after we’d successfully decreased waiting time, that there was some correlation between size and lead time. But we couldn’t see that until we’d addressed the wait time first. If your team’s work makes up only a small part of the total value stream, then your estimates will likely be poor indicators of when you can get your product.

### Explaining Upper Control Limits

A UCL essentially means that the majority of normal events are expected to occur below this limit. It reflects the degree of certainty for predictions. You can choose to be 95 percent certain ( $2\sigma$ ), 68 percent certain ( $\sigma$ ), or 50 percent certain (average, not recommended). As a rule of thumb, I select UCL at  $2\sigma$ , under which 95 percent of events are expected to occur.



A way to visualize UCL is to imagine a frequency distribution centered around the mean. The farther away from the mean we move, the fewer occurrences we can expect to find. Thus, UCL represents a cutoff point of the tail of the distribution.

Here, I won’t get into the statistical ways to calculate the UCL. One hack to estimate your UCL is to make a chart like the preceding one, with one bar for each lead-time

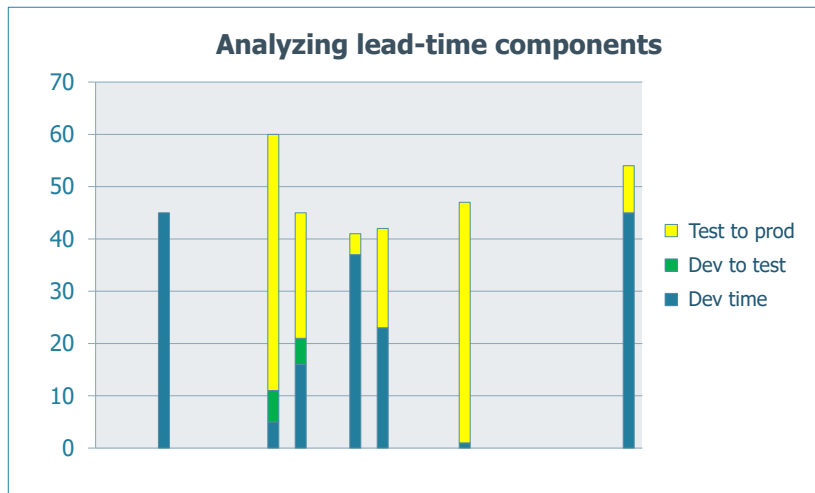
observation. Draw a line across the top of the bars, skimming above the majority of them but cutting across one or two. The level of certainty that you will hit this number is roughly equal to:

Number of bars above the line / Total number of bars

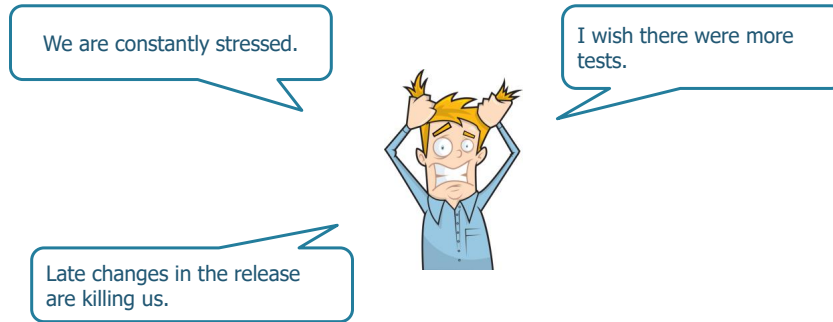
This is a rough and crude way to hack it, but it does give you a good enough approximation, especially if you're under time pressure, have little data, and need to make a call quickly.

## How We Located Our First Improvement Opportunity

Kanban makes it very easy to identify bottlenecks. We simply look for piles of tickets waiting. We didn't see a queue as clearly, though. One reason was that we had WIP limits across the board. The challenge was to understand where the opportunities for improvements might be. So we tracked the key components of lead-time data to see whether we could identify opportunities for improvement. Let's take a look at what the following figure can tell us.



So where should we start improving? We were leaning toward Test to Production as our biggest improvement area, but there were few data points, and the data wasn't conclusive. We walked over to the change management team—the team in charge of system testing and production updates—to ask their opinion of the situation. They were swamped with work, and we had found our bottleneck!



Change management was a team of nine people responsible for rolling out changes in 70-plus systems, ranging from 1970s technologies to modern ones. They had a lot to do.

The change management team had already begun taking steps to improve their situation. We decided to tackle the problem together with them on three fronts. First, we introduced Kanban in change management to help them prioritize their workload, gain time to get the quality right, decrease stress, and foster teamwork. Second, we stopped doing late changes to releases. We struck an agreement between development, change management, and marketing about the cutoff time for changes to releases and made sure everyone stuck to it. And finally, we engaged developers to add automated test scenarios to our system test environment. This simplified testing and, more importantly, tested feedback.

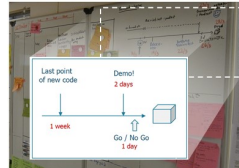
## How We Stopped Late Changes

We had several examples of late changes pushed in very late during the release cycle, which made it hard to complete our system testing. We backtracked through the quality efforts required and concluded that this usually happens one week before the release. That was the time required to complete our key system tests.

When we looked at previous releases, system testing was often only partially complete. Several reasons accounted for this: market and time pressure, late changes, and the difficulty for the person accepting/rejecting a late change to overview the current status of the release.

We needed to find a way to change our behavior—to build quality in instead of pushing quality out.

To address this, we added a timeline at the far end of the Kanban board for all the development teams involved (see the following figure). This included the cutoff time for changes to releases. And we asked the change management team to update this for us for each release.



You may be wondering, “Didn’t you have a process before?” Yes, we had a very hefty and well-documented process, dictating how and when things should happen. The process stated that no change could happen as late as four weeks before the release.

But advances in technology and different risk profiles for systems made people realize that some late changes were not as risky as others. Just because a process is well documented doesn’t mean that it describes how things really happen. This is one of the upsides of Kanban: by demonstrating how work happens, right now, your interventions are more likely to target real problems.

Basing improvement decisions on documented processes isn’t a good idea, because it’s a flawed premise based on flawed assumptions—namely, that documents reflect how work really happens and that they provide fact-based guidance to your biggest improvement opportunity.

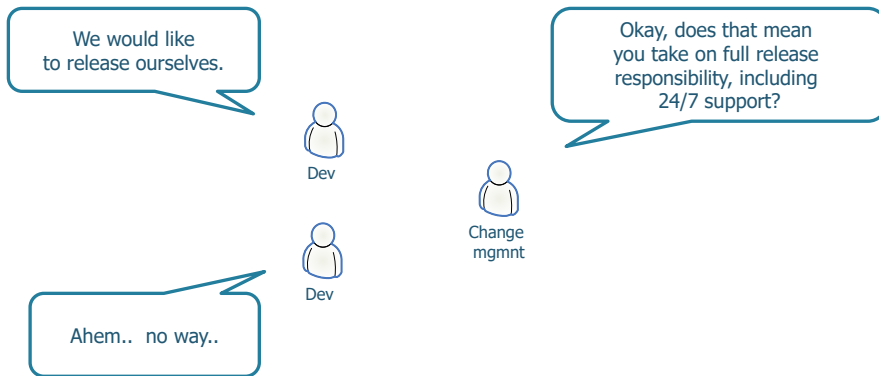
A well-documented process is unlikely to show how things really happen. Software development moves fast. If it’s perfectly documented, it’s probably already outdated. A better way is to base improvements on firsthand observations from people doing the work. Start by asking, “What could be made better or simpler?” Then validate by asking for real-life examples. Finally, use data over time to see whether the observed event is repetitive or a random event.

As we got more data, we saw that we could shave off a big chunk of the lead time if teams could release themselves—more precisely, if they could exempt themselves from the monthly release window. This has some advantages, such as we would virtually eliminate waiting time for system testing. Smaller releases mean that it would be easier to identify causes for quality problems.

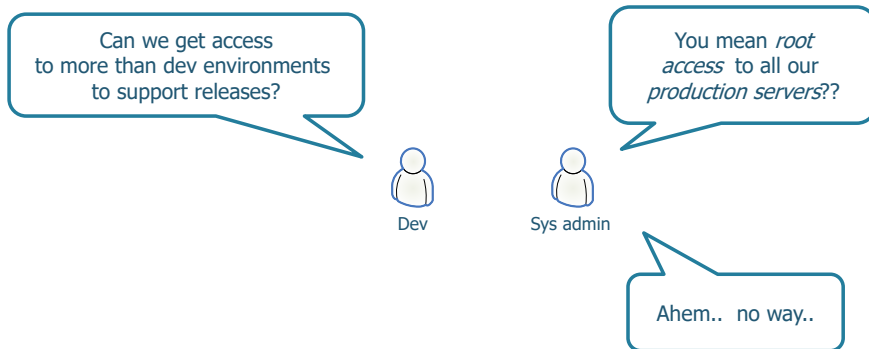
It would also remove the delay aggregation effect—such as what happens when a one-day delay near the release window turns into a four-week delay because of the monthly release window. If teams could release themselves and step out of the monthly release window, a one-day delay would be just a one-day delay.

Making this change may sound simple in theory, but it proved difficult in practice. The conversation between teams derailed quickly.

Let's take a look at a typical conversation between development and change management:



And this was the typical conversation between development and the system admin:



It was ironic, to say the least. We had a fact-based improvement idea, but no one wanted it. The problem was that our first approach was too blunt. It drove the involved parties to expect the worst and ignore the positive effects of our idea.

---

### The Importance of Communication

---



The importance of the role of communication during change cannot be overemphasized. Keep this as a rule of thumb: a change should never come as a surprise. When change happens across multiple functions, you cannot leave communication to chance. Verify if and how the message got through. Use examples to drive home your point. Avoid abstract terminology, because it could invite reinterpretations of your message.

---

After failing with our first approach, we changed tactics. We broke the idea into small, concrete steps that could be taken one at a time.

First, we let change management prepare a release checklist for all important steps necessary to move a release to production with high quality. This explicit checklist was shared with all development teams involved and helped clarify expectations of release work.

Then we added new roles with different access rights to test and production environments. This let developers perform simple forms of deployments and error investigations.

We also differentiated risk profiles. We decided it was okay to release at will for systems with few dependencies and good test coverage. We continued to release under the normal release window for systems with many dependencies and low test coverage.

Finally, we freed up time so that change management staff could spend 50 percent of their time working directly with the development teams to mitigate quality problems earlier in the development cycle.

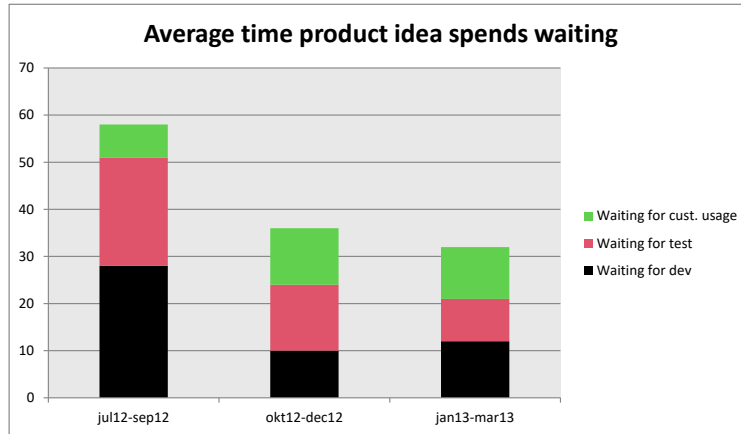
We moved through each step one at a time. It might not seem like a big deal that we finally got to a point where the development team was able to release outside the normal release window, but this would have been inconceivable two years prior.

### How Lead Time Improved

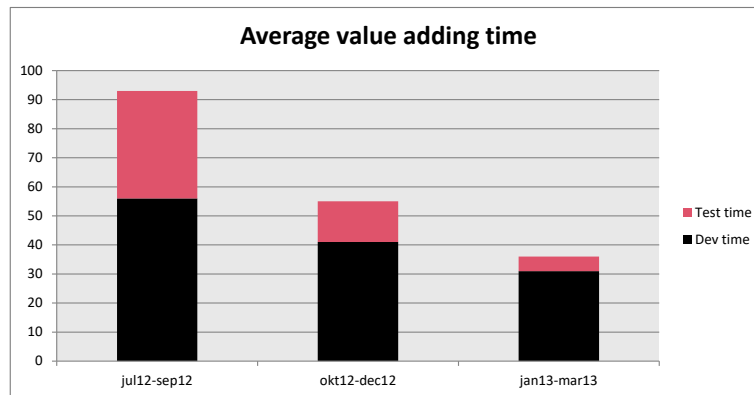
Did we improve? That's the most important question. All these changes don't matter if we don't have improvements we can point to. Let's look at some data, starting with lead time for released products, as shown in the [graph on page 38](#).





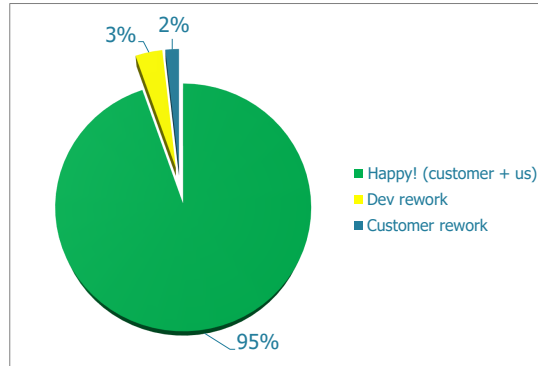


As you can see, the waiting time dropped, roughly by a factor of two. And you can see the same pattern for value-adding time separated into development and system testing in the following graph.



Several factors contributed to the drop in waiting time and value-adding time, including focus on flow, less rework, the ability to release when ready, and a better understanding of the technology being used. Less rework is related to better-prepared inflow, earlier validation, better test coverage, and the ability to mitigate the impact of late changes. The biggest change was in *time through system testing*, which we reduced by a factor of 7.

Just as we wanted to make sure that we were actually improving, we also wanted to make sure we were shipping things that had value to the customer—things that could actually be sold. The pie chart shows how we performed in this regard.



We learned about value by asking customers and end users after a release whether they were happy with the delivery. As you may recall, we recorded the results at the end of the Kanban board under two sections: Popular and Oh Crap! Something could fall into the Oh Crap! category in two different ways: either the customer rejected our delivery (customer rework) or we didn't ship the product because we weren't happy with it (dev rework).

---

#### Have a Feedback Loop

---



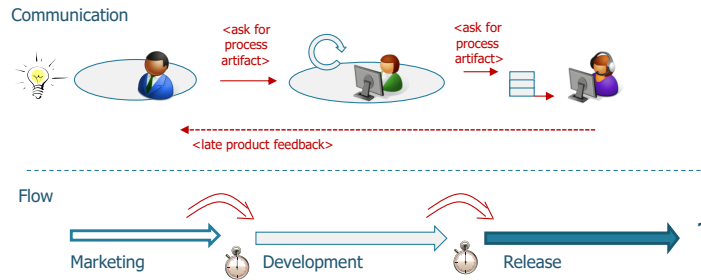
While we're still learning the discipline of collecting this data (it's easy to forget to call back after first usage), our data does not prove our hypothesis that we were capable of shipping things of value. But what's important is that we now have a feedback loop in place to learn from.

---

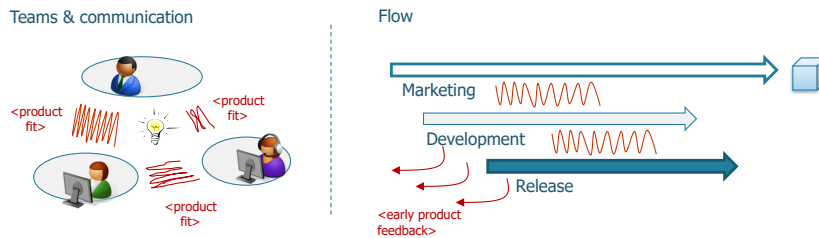
## Comparing Now and Before

The numbers tell a story, but we wanted to make sure that the people themselves felt as if things had changed. As you may remember, there was originally a sense throughout Company H that improvements had stalled. Were people feeling better about the changes at Company H?

The figure [shown on page 41](#) illustrates how people perceived communication at Company H before Enterprise Kanban. Communication revolved around completion of required process artifacts as defined by the individual function. Holistic feedback was returned late, either at system testing or when the complete product was in production. Sprint commitments were geared toward subsystem changes. The quality of the final product and the progress of product ideas were hard to grasp.



The following figure shows how the changes resulted in a dramatically different situation. Today, we have the first feedback on customer value (if we understand and can communicate it) before the product idea is shared with development. Validation of product fit—understanding whether the solution meets most prioritized expectations—happens continuously.



Today, conversations tend to be focused on what value we want to create and what problem we need to solve. Progress happens when the quality, not the time, tells you when a product is ready. While the basic organizational structure hasn't changed (we're still organized in multiple teams and in functions), both behaviors and conversations have shifted dramatically. The people involved will tell you when something is ready to move forward. And we trust them to make that call.

"We talk a lot about customer need and the value we want to create today."

—Manager

We managed to reach a 2x improvement in lead time over a period of 18 months. Kanban didn't make this happen; the people who worked on the projects did, along with their managers. What Kanban helped us do was to make visible how things really work so that our processes can be adapted on the fly.

It's also interesting to note what we didn't do. We didn't improve by marking ourselves against a certain process method (for example, how agile are you?) or organizational model. We looked at real observations from our end-to-end flow to decide what to improve on next. And we stuck with it until it was done.

The tricky parts were dismantling old processes and routines as we discovered better ways to do things.

## Make Your Own Improvements

You may find yourself in a similar situation where you want to improve your workflow end to end. Or you may want to apply some of the things we've just discussed to change how your department functions. As I've just mentioned, the challenge was in dismantling old processes and routines as we discovered better ways to do things. Here's some advice to help you overcome these challenges:

1. Visualize the whole value stream. Engage management in improving end to end, not just a small part. It's when we see the entire customer journey that we can unlock the greatest potential.
2. Try making changes by applying small experiments. Everything is new the first time. If your first idea falls flat, try something smaller, but don't postpone the journey. Run small experiments to learn whether they work.
3. Talk about a change. Listen. And talk. Treat people as thinking beings and suggest better options. It's possible to change old routines and habits even across organizational borders if you persist in talking about them for long enough. The greatest form of respect is to help someone evolve.

## Next Steps

Now that we've seen how it's possible to improve a complete value chain, let's take a look at how to solve a bottleneck under heavy pressure. How can you improve life (for starters: keep your nose above water) for a small team handling 350+ systems across multiple tech stacks? Let's hear the story from the Change Management team.

# Using Kanban to Manage Change

---

Unfortunately, it's easy not to see problems even if they're right under our noses. In this case study, we'll look at how a company's change management team adopted Kanban. Using Kanban, managers were able to eliminate stress, build an effective team that took responsibility for the big-picture goals, and work with other teams to address behaviors that were causing recurring quality problems.

The change management team was responsible for tracking various requests from internal developers, system owners, and third-party vendors and grouping them into appropriate releases. The team was under pressure to release code more frequently and felt they didn't have time to thoroughly test beforehand. Let's see how the company improved its process and avoided burnout by introducing Kanban into change management.

## The Challenge: Managing Dependencies Without Burning Out

We're still looking at Company H, the company we looked at in [\*Using Kanban in the Enterprise\*](#), but now the focus is on the team in charge of releases and production updates. The change management team accepts change requests from internal development teams, system owners, and external vendors (for upgrades, for example).

The team managed changes across 350-plus systems, ranging from standard systems by outside vendors to custom in-house boxes. The technology stack ranged from old VMS systems to Java. Many of the production systems were parts of the country's critical infrastructure. Maintaining knowledge and dependencies across such a diverse technical stack was a huge challenge for a team of just eight people.

The team had been working under a lot of stress for a long time, and the work overload meant that they had little time to test code or perform quality assurance testing. At the same time, both internal development teams and outside vendors were requesting releases more frequently. The team was overwhelmed.

Before adopting Kanban, the team managed their work with a ticket system. Each team member got requests for a release, patch, or system change as a ticket and was responsible for making that change to test and/or production systems.

The team's manager, Birgitta, described how bad things were before Kanban. She was one of the key figures instrumental in pushing the changes through.

### **Why We Needed Kanban**

*by: Birgitta, head of change management*

*Q: Before you started, what did you think Kanban could help out with?*

Most importantly, visualization, for everyone in the team to see what was going on. This was a big difference from before. When we were using the ticket system, we had little or no overview of what we were really working on.

Secondly, to be able to see what we were spending our resources on and to get a common forum going where we discussed that, like a stand-up meeting. Last but not least, we also wanted to improve flow through our team.

*Q: At what point did you think, "This might work!"*

When we started, we had a very rough and chaotic board. We had tickets piled upon each other and a poor overview. At that point we said, "We need to fix this," and we started to improve the structure and overview of the board. We also started asking questions about if work really was being done. It turned out that a lot of work that was piled up on the board was not actually being worked on. I remember one occasion when we found a team member with 20 active items! We asked him, "Are you really working on 20 things?" And the reply was, "Hmm, actually, no." So we decided that the board should reflect active items only and told everyone that if something was not being worked on, it was no longer active, and we started putting the items back into the queue. That made a difference. The board got cleaned up. The team started to take responsibility for the tasks at hand, we got focus, and the stress decreased considerably.

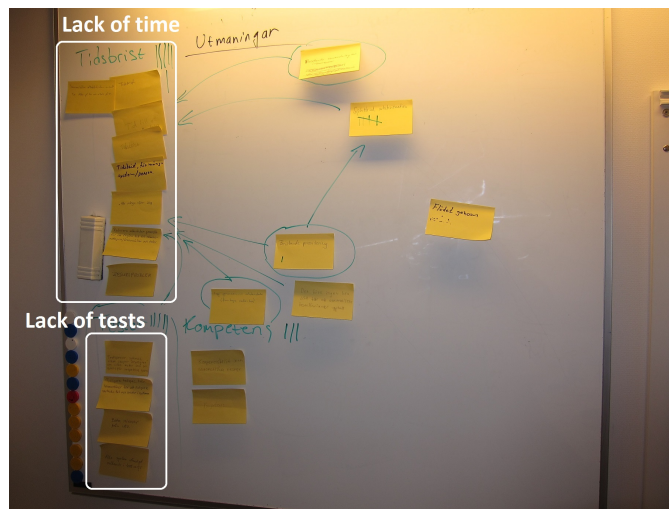
## **How We Got Started**

It began with two managers observing how Kanban was used in software development to visualize and track work across multiple teams. This inspired them to take a Kanban training class to see if Kanban could be helpful for them. On the train home back from the class, they sketched their first Kanban system and agreed that it would help them get an overview of the work they

were handling and enable them to work together as a team instead of being on their own.

I was invited to facilitate a kickoff workshop on Kanban for the change management team. Because we wanted to start with the challenges and get the team's perspective, we began with a small retrospective meeting. We asked each team member and manager to list the top three challenges on Post-its, which we then grouped on a whiteboard. We separated causes from effects. Finally, we asked the team members to vote on what they believed was their most pressing concern.

This exercise allowed the team members to share their perspectives on the situation and to agree on what their key challenges were. We were now ready for a discussion on how to address them.



As shown in the preceding image, two key challenges were identified at the workshop: lack of time and lack of testing during releases.











We discussed examples of how other teams use Kanban and the pros and cons of each approach. We asked the team to use simple thumb voting (see [how to thumb vote on page 13](#)) to tell us whether they would be willing to give Kanban a try. Once the team agreed, we could put up the board.

Let's look at how we set up the Kanban board for a change management team.

## How Our Process Worked

Change management needed to see an overview of the current active release (in the Current Release lane) as well as upcoming change requests (in the Not

Assigned lane) on the Kanban board. Each lane had its own priority: tasks higher on the board had higher priority. Notice the Follow Up section on the board—that's where services are verified in production after changes are released.

Release	In queue		Ongoing				Done	
 High prio								
Releases   Small Medium   Liquidate Coord.	Current release	Content	Admin	Prep.	In test	Put in prod	Follow up	
	Not assig.							
 Improvement								
 Patches								
 Std. change								
 Problem								
 Other								

Let's take a closer look at how work flowed through the board. Each lane represented a specific priority and demand type. Tickets in each lane had a specific color to make it easy to visualize what was happening. We could see progress on non-urgent demands, such as improvements.

Prio	Name on the Board	Description
1.	High prio	Critical production fixes
2.	Releases	Scheduled release updates to production. Runs on a four-week interval. Colored categories of release requests are: <ul style="list-style-type: none"> <li>• Small—small changes</li> <li>• Medium—medium changes</li> <li>• Liquidate—remove from production</li> <li>• Coordinate—application change requiring explicit coordination</li> </ul>
3.	Improvements	Improvements run and owned by the team





back in the prioritized queue, or they'd get that item done. We quickly learned that the team was carrying a lot of passive work, which was stealing energy and focus.

---

#### Focus Your Board

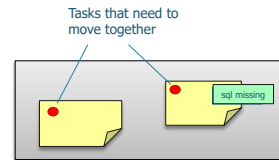
---



In the beginning, we envisioned that we would display up to four upcoming releases on the board, but we quickly found that was impossible. The overview was lost. We switched to displaying two releases (current and next), and this worked much better. Narrow your board's focus.

The board felt messy even after a month, so we cut the Post-it notes attached to the board in half. This reduced the size of the tasks and made it easier to see the overview. On each Post-it, we tracked the name of the change and the ID number of the corresponding task in the ticket system. This let us identify and track tasks while keeping a neat board.

We also added colored magnets to show which tasks were blocked and to show *tags* explaining why the task was blocked. Dots showed which changes had to be coordinated with the upcoming release and which changes could be released individually or at a later time.



That made a huge difference. The work was largely distributed across different team members. The lack of an overview meant that no one could really make a snap decision about whether to move a change out to production without explicitly coordinating with the rest of the team. Now they could.

---

#### How You Know Your Kanban Board Works

---

When you look at a Kanban board, ask the following four questions to see whether the board is working:



- *Overview.* Can I get an overview at a glance? Can people working with the board see what to focus on, what is prioritized, and whether any items are blocked? This helps team members see what to focus on and managers see where to invest improvement efforts.
  - *Transparency to progress.* Does the board help people track progress? By enabling people who depend on the team to see
-

---

### How You Know Your Kanban Board Works

---

progress, you allow them to coordinate work proactively on their own, without having to bother the teams for status reports.

- *Meaning.* Can the team describe the meaning of the visual indicators on the board? The point is, if no behavior is associated with the visual indicators, then they have no meaning.
  - *Usefulness.* Are conversations happening in front of the board? If the answer is yes, the board works well as a catalyst for insight and decisions.
- 

Let's look at how we engaged with the board.

## Working with the Board

Each team member was responsible for their day-to-day list of tasks, but team members still needed to work as a team on the combined workload. To balance this, change management had two stand-up meetings a week where they shifted from what they needed to do as individuals to what they needed to complete as a team.

The team ran a stand-up meeting in front of the board on Tuesdays and Thursdays. The meetings were short—a maximum of 15 minutes—and they were run by the team's managers. The agenda was to walk the board, reviewing whether something needed attention. At this meeting, the team also solved problems and clarified who was responsible for following up on blocking issues. Finally, the team reviewed the Prioritized queue (which was kept to the left of the Kanban board) so that the team had an idea of what was coming up and what was important.

The management team prioritized and pulled requests from the ticket system and added them to the Prioritized queue. This helped the team keep track of work in progress even if the workload increased. Smaller requests coming directly to the team members were reviewed during the twice-weekly stand-ups.

It's worth noting that when prioritizing, the teams reserved the right not to take on a task, in which case the stakeholder would be given the option either to put the task back into the queue for later or to solve the issue on their own. We tracked the number of *won't do* requests over time.

---

### Expect Your Kanban Board to Evolve

---



It's natural for a Kanban board to evolve many times during the first two months of operation. Items on the board such as WIP limits, lanes, and work states will inevitably change. This is natural. In the beginning, try to update the layout once a week. The goal is to maintain a clear overview for all to see.

---

Because this team acted as the company's second-line support, answering application-specific questions and troubleshooting were part of the daily workload. Soon after we introduced Kanban, we experimented with having just one person field support questions during the week and freeing up the rest of the team to focus on other assignments. We called this role "Today's Backo" (back office) and rotated the assignment among all the members.

We struggled with this policy at first because people tended to approach the person they were used to working with instead of the week's designated Backo. To solve this problem, the team suggested posting the name of the Backo on the Kanban board. This way, anyone looking at the board would know instantly who to talk to. We added a separate column on the far right of the board to list the contact person's information. This was tremendously helpful in getting people to go to the correct support person.

We had another challenge: the uneven distribution of skills. Some people didn't have the experience necessary to be able to handle some of the more frequently occurring requests. The Backo person first had to try to find a solution to the problem before getting help from the rest of the team. It was important that the team member helping the Backo did not directly solve the problem but instead guided the Backo toward finding the correct answer. This made the support issue a learning exercise.

---

### Handling Common Issues

---



It's a good idea to maintain a FAQ on a wiki for common issues. After a couple of turns, the support person should be able to answer questions without pulling in help. You don't need cross-functional skills on all matters, just on frequently recurring support demands.

---

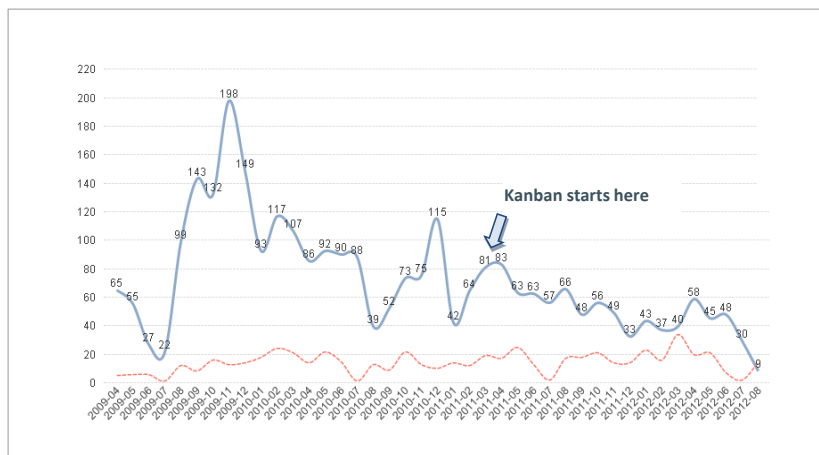
Keeping the ticket tracking system and the physical Kanban board in sync was a challenge. The physical board was tremendously valuable because it gave the team an overview of what everyone was working on. Team members working on their own also felt connected to the rest of the team. The board triggered conversations and helped members share responsibility for the





moved from zero to a small but steady trickle? This showed us we had a healthy balance between short- and long-term activities.

Let's take a closer look at the metrics we gathered. The flow statistics were helpful in validating the effect of changes over time. The Kanban board should be treated as a snapshot in time and the flow metrics as a way to see seasonal effects and trends. The following graph shows lead time for the change management team.





The fixed blue line denotes lead time, and the red dotted line denotes the number of tickets the team received. The time to resolve a ticket dropped from 60 days down to roughly 30 days once the team was up and running with Kanban. Seasonal effects may have influenced the results, and we need more data before we can completely rule seasonal effects out. However, the drop in lead time does correlate with the observations from the Kanban board along with the decrease in the team's stress level.

Now we've seen the metrics, let's take a look at the lessons the team learned.

## What Lessons We Learned

After running the board for a while, we realized that how the board looked and felt was important. Was it clean, neat, and tidy? Did it communicate relevant things, or were there too many irrelevant or extraneous items? If the manager was meticulous about maintaining the Kanban board, the team took pride in that. The resulting work reflected that pride, and the members started working well as a team.

---

### Actions Inspire Teams

---



Keep an eye on your board. If you, as the manager, keep the board neat and tidy and act on what happens, you'll inspire the same behavior in your team.

---

We reviewed the board after running Kanban for a year and made an important observation: the visual overview helped managers spot patterns, such as identifying the less mature system providers. With the board, managers could see at a glance which providers had low control over product quality because those providers consistently released patches after big releases. Managers could also flag providers who submitted multiple patches instead of bundling them together. These things had gone unnoticed before. Kanban has helped managers learn how their providers work.

Let's hear what the team had to say about the experience.

## Comparing Now and Before

Was Kanban helpful? The simplest way to find out was to ask the team members whether they wanted to continue using Kanban. We learned that Kanban helped the team develop a single picture of what was happening and what to focus on. Everyone saw the same overall picture even though they spent half of their time working with other teams. Kanban helped the team find and maintain a sustainable pace even when the workload and demands on the team were continuously shifting. So, the answer was yes.

The biggest noticeable difference was the team's stress level. While the team still had plenty to do, they now felt more in control of what was going on. The Kanban board gave them a better overview of what was happening, what state the work was in, and what items needed attention. Interestingly, this has been reflected in the conversations at the stand-up meetings. Before, the conversations were chatty; today, they're more focused, and the team now talks about deviations and what they need to act on. This is because the team feels they're in control of the day-to-day work, making it unnecessary to bring up these items.

When we asked the team if they were seeing the results of their efforts, this is what we heard: "Before, we felt like we spent a lot of energy that just disappeared into a black hole. Now, if we move tickets forward, it actually makes everyone feel that we are taking one step closer to completing the release."

We experienced an unexpected side effect of using Kanban. Adjacent teams, notably the system admins, began attending change management's stand-up meetings. When the system admins saw change management's Kanban board, they said, "This stuff is important to us. We've been invited to the development team's stand-ups before, but we felt that they discussed items that were rarely relevant to us. Your stuff is more relevant. Can we join your stand-up?" Incident management also joined because they were interested in the status of various incidents. They also shared information about the prevailing mood across other teams. Incident management provided us with a *fingerspitzengefühl* (literally, "fingertips feeling") on other teams, letting us know if they were worried or in trouble.

Team members allocate 50 percent of their time to working with other teams, such as helping development teams prepare and test releases earlier. That would have seemed impossible just a year ago, before Kanban.

## Make Your Own Improvements

A simple rule to follow if you don't know whether a tool is right for you is this: make sure you have a good reason for using it. "It sounds cool" isn't enough of a reason. A team retrospective is an easy way to find out what difficulties and challenges the team is struggling with. If what you hear correlates to what Kanban can help with, then you've found your tool.

The managers of the team in this case study saw other teams successfully using Kanban and attended a Kanban training session to learn how the tool could benefit the team.



In times of stress, it can be difficult to know what areas need improvement or to identify areas that would benefit the most. Here's a quick guide:

1. Focus on active work. When you look at your board, what do you see? Remove inactive work from the board. Either complete it, remove it, or put it back in the queue. Make sure that what's on the board is what the team is working on.
2. Know what will get worked on. As the manager or senior member of the team, you need to take charge of prioritizing all work that comes to your team. Don't forget, this includes communicating with stakeholders who won't be getting their tasks completed right away.
3. Quality first. Getting better isn't something you do once in a while. Free up time to make small but consistent improvements. Get the quality right first, and the flow will come.
4. Celebrate! Recognize the team's accomplishments and individual member contributions.
5. Help other teams.

## Next Steps

The Change Management team improvement journey demonstrated to me that Kanban, together with leaders committed to making improvements, makes a difference in a DevOps context.

But how do you improve a larger software delivery of a core business system late in the process when delivery has fallen behind schedule and with a customer that is losing faith? Let's hear the story of how Kanban was used to save a derailing project.

# Using Kanban to Save a Derailing Project

---

The first step in solving a problem is seeing that it exists. That sounds simple enough—except in situations where everyone wants to solve a different problem. Everyone involved knows something is wrong, but each person sees different things. It's really difficult to figure out how to fix the problem if no one agrees what the problem is in the first place.

Throw in other challenges, such as a looming deadline and lack of coordination across teams, and the problem seems unsurmountable. Let's see how a company used Kanban to identify—and agree on!—the problems, prioritize the issues, and come up with a plan to address them.

## The Challenge: Restoring Trust by Solving the Right Problem

Company F, an open source platform provider, was six months into an eight-month project when the client threatened to pull the plug because of a series of problems. This would've meant saying goodbye to a key client who was considered important to the company's operations. What had initially run well—a pilot delivered on time with a happy client—had turned into a growing pain of overtime, rework, and technical debt. After several scope adjustments, the final product to be delivered was the bare minimum the client needed. Nothing was left to trim, and it looked like the team was still going to miss the delivery date.

On a positive note—and there's always something positive—the development team consisted of very committed members and a positive project



---

**In situations like these, you'd be well advised to equip yourself with proper protection!**

---

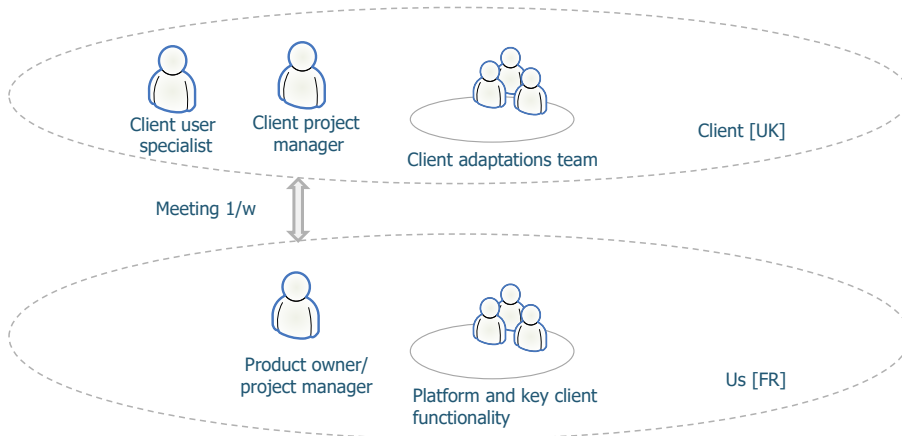
manager/product owner. The team had strong backing from management and weekly conversations with the client. The dialogue wasn't always pleasant, but at least both sides were talking regularly.

We discovered the first problem right away. When I asked each person to identify the biggest problem, everyone had a different answer. Everyone had their own view of the situation, and no common view or shared perspective surfaced.

The project manager saw the team struggling to deliver quality products, getting overstressed, and consistently underestimating the level of effort and time required. For new tasks, the estimates were as much as five times off. The development team felt there was too much switching between tasks and the members were doing too many things at the same time. The client felt the team was in way over its head and wasn't up to the task of completing the project.

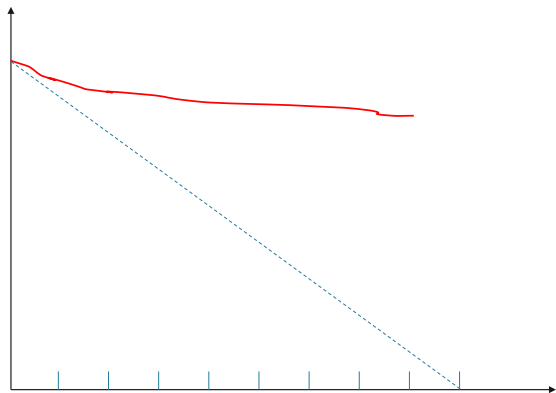
As you can see, the answers varied greatly depending on who you asked. Interviewing everyone involved didn't help us figure out what to address first, so we looked at how the team was working together.

Two development teams were working on this project, as illustrated in the following figure. The foundation and the main bulk of the work was completed by Company F's team in France, while the client made its adaptations through a team in the United Kingdom. The two project managers met once a week to revise priorities and to review progress.



Company F's team had been using Scrum for a couple of months, and the project manager was a certified Scrum master. The team didn't think Scrum was working. Apparent were several signs of trouble, such as the team

working massive amounts of overtime, often until one in the morning. You can see the worrisome pattern in the sprint burndown:



My first thought when I saw the burndown was “why didn’t this trigger a reaction?” Let’s look at how we regained control of the team’s time.

## How We Got Started

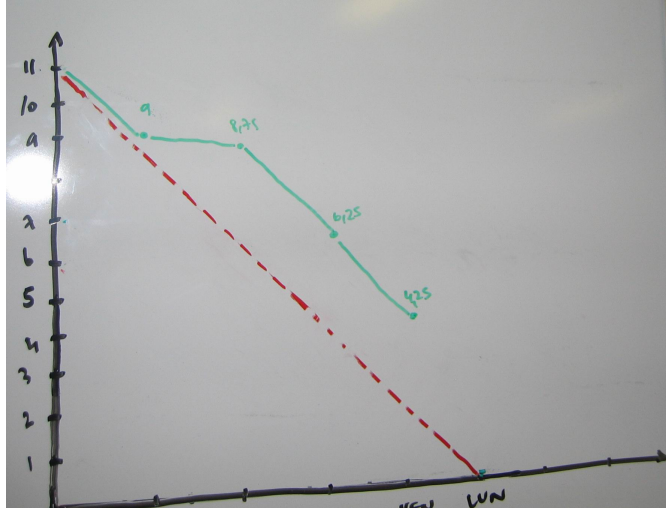
The decision to try Kanban came from Company F’s CEO. The team didn’t know about Kanban until the first board went up on the wall (see the following image). The board listed all work-in-progress items under the Dev, Merge, CI, and PO Test columns. Urgent items went into the Fast Lane at the top. We allowed only one item in the Fast Lane at a time.

Backlog		Estimation		Runnable @ client		
	In queue	Estimated	Fast lane	In work	Done	
			Dev	Merge	CI	

We didn’t change anything else in the team’s work procedure or process. The team continued to run two-week sprints, but with this board. The main

difference between the new Kanban board and the previous Scrum board the team used was an added limit on work in progress. We talked about WIP limits, which each team defines for itself, in [Leading with Kanban](#). The new board also made the value stream more prominent.

As you can see in the sprint burndown chart, after two weeks little things began to change. There was an actual burndown, as opposed to the flatline we saw on previous charts.



Kanban brought a shared overview of flow and a work-in-progress (WIP) limit to the team, and the team could see positive results after just two weeks. The team worked on fewer stories at a time and was able to both develop and test the features. We saw the team could produce working software if the conditions were right.

## Develop a Shared View of Progress

We realized the team and the project manager were tracking two different things. The project manager was tracking what activities were completed and checking the project plan. The team tracked the progress of the current sprint. Because the client, the project manager, and the team were looking at different things, they couldn't view the project's overall progress in the same way. Kanban provided a way to see all the steps needed to move from the customer request to executable software. Visualizing the process also helped flag problem areas.

Before Kanban, producing this overview would have required extensive discussions with multiple people and compiling data from different tools.

### Getting a Shared View on Progress

You can develop a shared view of the team's progress in many ways.

It doesn't matter which indicator is being used, as long as it's meaningful to the people involved.

Here's a tip: if the indicator being used to report progress isn't being used to make decisions, pick something else that's more useful.

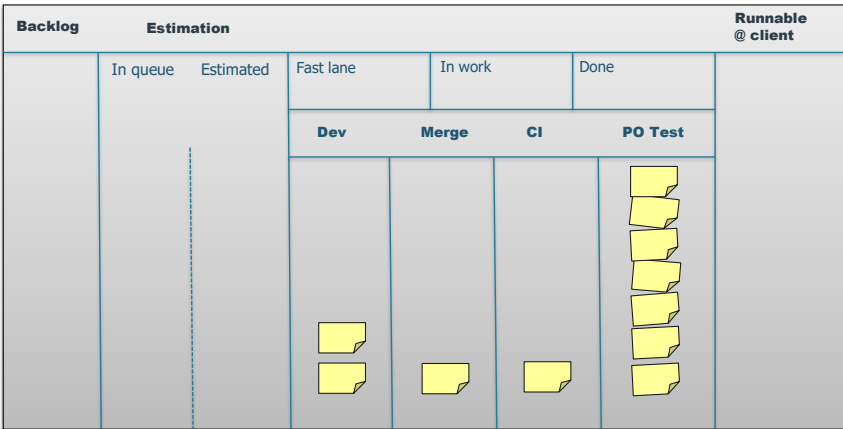


If you don't have a progress indicator you trust, take a quick poll of the project participants' confidence level about making a deadline (on a scale of 1 to 5, with 5 being very confident). Yes, it's a crude measure, but it's better than nothing and better than using a bad indicator. If you include a cross-section of the people involved in this poll, the results reflect the team's current best knowledge.

Or you can use a simple release burndown that's visible to everyone. Make sure everyone agrees that *done* in this context means *tested*. Have testers handle progress reporting to keep this honest.

### Figure Out What to Fix

Two weeks in, we uncovered more interesting patterns on the Kanban board. Stories frequently got stuck in the testing phase. Other stories on the board would be removed after the sprint even though they were incomplete. Even more worrying, those stories resurfaced on the board a few sprints later.



The PO Test column showed our first bottleneck (as shown in the preceding image). The team was quick with development but struggled with testing and quality assurance. It turned out the project manager handled acceptance testing, but they could spend only half of their time testing. At the end of the sprint, many of these features shipped even though testing wasn't complete.

A part-time person testing code from four developers—no wonder we had a bottleneck!

We found that some of the stories were far too complex to complete coding and testing within a single two-week sprint. Because they were too complicated, these half-completed stories were removed from the board. But because they were crucial to the overall project's success, they would come back on the board. The rework was affecting team morale.

Releases were also taking too long. The team would learn about a detail that needed to be tweaked while preparing the Monday release. The release date would shift a day or two to accommodate the change, requiring the team to redo some of the prep work. We found the team was spending two to three days on release work instead of just one day.

## Make Changes in Workflow

When you want to change a project stuck in a downward spiral, you need to take a stand. If it works, trust will build, and you'll gain a little bit of slack, which you can use toward long-term improvements. You will slowly work your way back up instead of going down.

Making a stand for quality is always a good bet. Our line in the sand was that we would include only stories of acceptable quality in each release, regardless of what was previously promised or what people were expecting. We also said release dates would always be on Mondays and would never shift. It wasn't easy convincing a nervous client, so we moved away from a two-week release cycle to a weekly one. That way, if we missed the expected release date, the client had to wait just one more week to get the desired feature.

With this decision, the team could focus. They did less emergency patching, spent less time on re-estimating tasks, and spent less time on repeated acceptance testing. Stories stayed on the board until they were complete, regardless of how many sprints they took. Instead of chasing things that didn't turn out as expected, the client focused on validating the quality of the releases. A faster learning cycle resulted because the team was getting feedback once a week.

We saw a boost in the team's morale the first time it successfully shipped a story that was too big for one sprint.

Making the shift wasn't easy, because it meant saying no to the client a few times when the feature wasn't good enough to be released. But we bet that shipping features that worked as expected would grow trust and put us on the positive upswing.

We had no shortage of ideas for things we wanted to improve, but an immediate one demanded our attention: we needed time to make improvements. This project was already late, and the team's schedule was jam-packed just to complete the essentials. We needed to find slack.

We found it in different places. One was in how the team approached estimates. Instead of spending a full day coming up with estimates for each story, we tried using the T-shirt sizing scheme (small, medium, large). The sizes corresponded to time: small meant one day or less, medium meant one week or less, and large was anything that would take longer than a week.

The first time we did this, we discussed the stories over lunch. By the time we got to coffee, each story had an updated estimate. By changing how we did estimates, we freed up time to work on other things.

---

#### Finding Spare Time During Periods of Stress

---

A trick to free up time is to look for planning efforts aimed at future work. The more distant the future, the higher the chance the work will be rendered useless by later changes and your preparations will have to be redone. So if you have no slack time for improvements, scaling down time budgeted for future work is a good bet. It's what you deliver that counts, not what you're starting.



For any estimates, start by clarifying the question you need to answer first. Then proceed with the simplest possible estimate that gives you that answer.

A very simple and effective technique to estimate a bunch of stories in one shot is to gather your team and silently sort the stories on a table, from the highest complexity to the lowest. Then estimate the story with the highest complexity, the one in the middle, and the simplest story. Any other stories in between will get their estimates relative to these three anchor points.

---

In this particular case, the new estimates didn't have a purpose other than to keep the plan updated. We knew we were late—updating the plan wouldn't generate any new information or insights. I was tempted to use a guerrilla



technique and just randomly pick estimates. We used the time we reclaimed with our T-shirt scheme to figure out where we needed the most improvements.

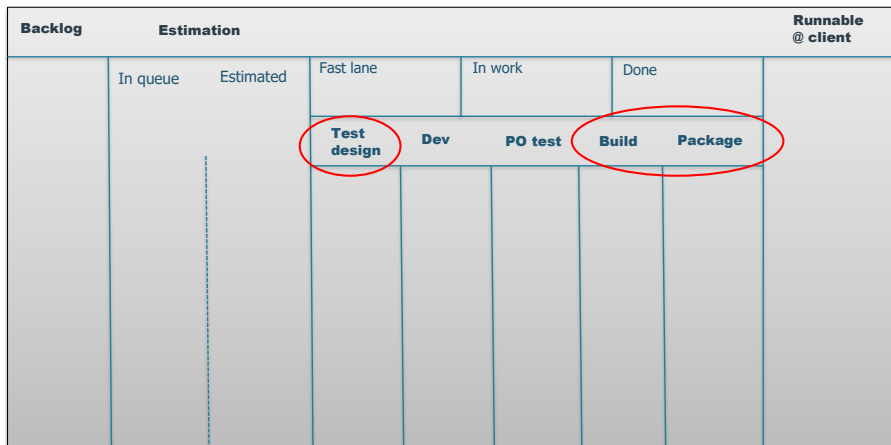
## How Our Process Worked

We kept making new discoveries about how things really worked, what processes we actually wanted, and how we should communicate the changes to everyone involved. We were constantly in a state of flux for the better. We even found ourselves changing our processes at least once a week.

We wanted to improve our testing and address the imbalance between our development and testing throughput. We looked at test-driven development (TDD) to help us be smarter with manual testing, decrease regression testing overhead, and give us readable code. Our team members came in an hour early for four mornings to attend TDD training workshops. Because the goal of these sessions was to make the team feel confident that they could change any part of the codebase, two of the training sessions focused on using TDD techniques with legacy code.

The team refactored the code after the workshops. They had known for a while that the codebase needed it, but now they were fixing the issues as a team and not as individuals.

We updated the Kanban board with two new columns: Test Design and Build/Package (as shown in the following image). Test Design made the choice between manual or automatic testing explicit. The Build/Package columns were used to coordinate release changes with the client team in the United Kingdom.



We knew that TDD couldn't be used to develop every single part. For example, the code for our graphical user interface (GUI) didn't have a stable testing framework. Although we had some ideas on how to fix this, implementing the necessary infrastructure would be time-consuming.

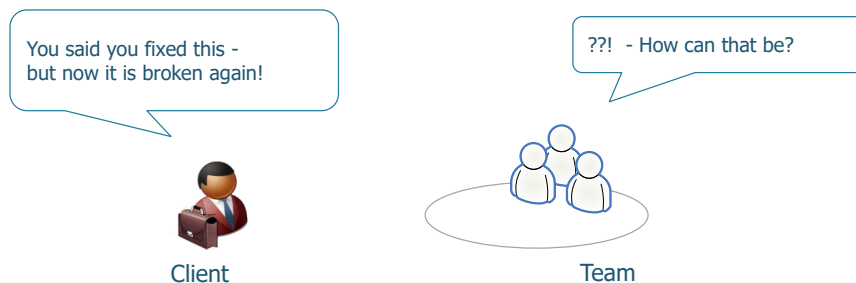
We compromised by focusing on baby steps. We designed for how we would test, and we used TDD wherever we could. We conducted manual testing where we couldn't write automated test cases. And finally, we committed to take one step forward each week toward implementing a working GUI test framework.

Even though the pressure was still intense, we saw small signs of improvements. We heard comments such as “why do we have things like this in our code?” and team members took the initiative to refactor code and fix quality issues. After a client meeting, the project manager said, “You know what? Even if they're still stressed out, they now trust me when I say we're going to deliver something.”

These small indicators showed that we were on the right track.

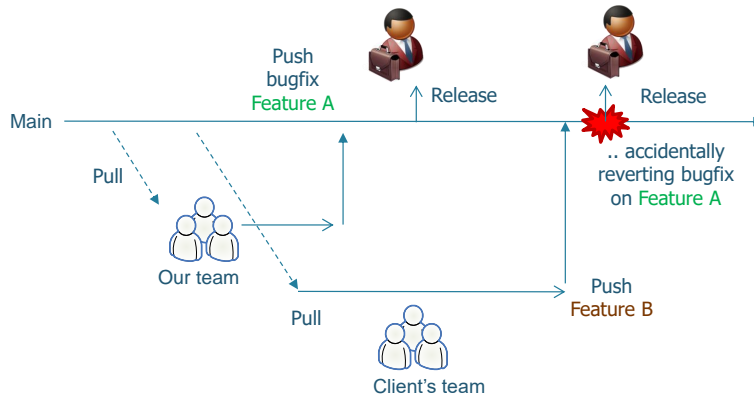
## How We Addressed Problems

After a couple of working releases, we were feeling confident. Then, out of the blue, we got a call from our client that a feature we'd recently fixed had suddenly stopped working.



After some investigation, we realized our bug fix wasn't the problem. It was a regression error because a U.K. developer had inadvertently reintroduced the bug by committing old code (see the [figure on page 66](#)).

It was clearly time to change how we worked with different teams. To prevent this situation from happening again, we created team branches and made it



clear that committing code to the team branch meant it was ready for integration testing. We moved the test suites to each branch to validate the code before committing to the main branch. Code in the main branch meant it was ready for release. Finally, we made sure that each team branch was automatically updated with the latest release on a daily basis.

This is a case in point where you learn about how work is actually done under high pressure as opposed to how it *should be* done as documented by some process or procedures manual.

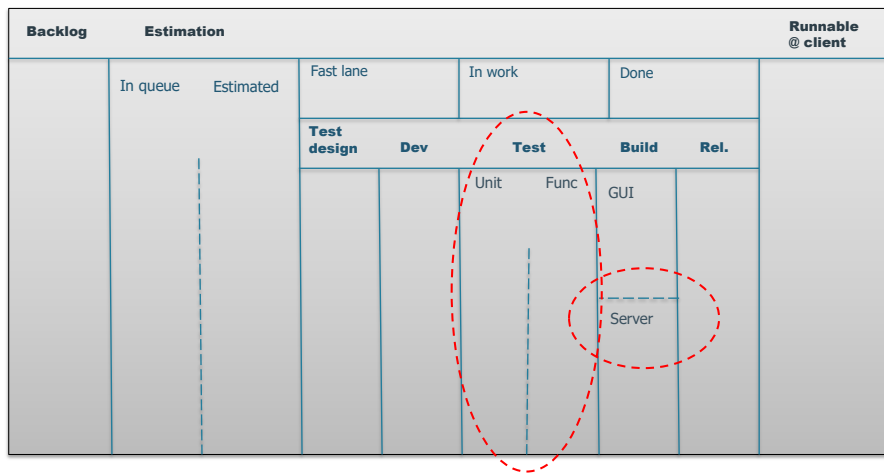
To improve, we quickly realized that we had to get the other team on board with our changes. After all, we committed to the same codebase. The first step was to have the two development leads check in with each other on a weekly call. We also kicked off a developer exchange program where developers from other teams came and learned how we worked. We covered topics such as branching, TDD, design patterns, integrated development environment (IDE) setup, continuous integration, and testing environments. After all the developers went through the exchange program, we were confident we would be able to work together as one team to maintain a higher standard of quality.



**The team, working together to maintain higher standards.**

## How We Continuously Improved

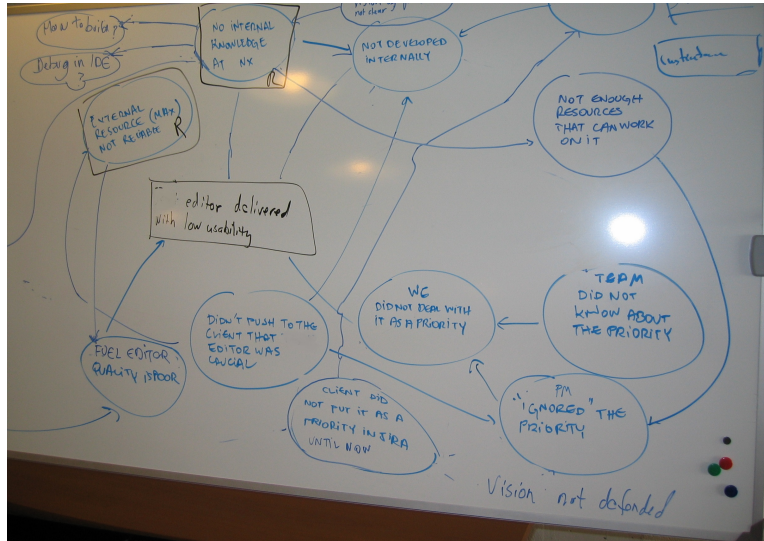
Everything so far was on the defense as we fought problems as they cropped up. After a few weeks, we saw team members kicking off their own refactoring initiatives and taking control of the board. Every change they made in their processes was reflected on the board. For example, as shown in the following figure, the Test column was split into two to reflect manual testing. A triggering mechanism in the Build column indicated whether a partial (client) or a full build was needed. The Kanban board became a living process and galvanized the team's commitment for each change.



We don't always have all the information we need at once in software development. To get to the bottom of a complex problem, we need to involve people with different perspectives. Getting everyone into the same room and in front of a whiteboard is an effective way to get to the crux of the problem quickly.

After a streak of successful releases, we ran into trouble with one. After some discussion, the CEO concluded, "I think this happened because the vision for the product wasn't clear." This could've turned into an argument. Instead, the team and the CEO gathered around a whiteboard and wrote down all the factors that could have caused the problems (see the [image on page 68](#)). It didn't take long for us to identify a set of factors that had been flying under the radar so far, the most pressing one being the lack of knowledge within the team to update an unstable third-party component vital to the product's use.

We decided to take ownership of the code, learn it, and refactor the unstable parts.



During times of stress, it's easy to squander your energy on the wrong things. If we had gone ahead with the CEO's initial thought to "fix the vision," we wouldn't have been able to improve in the few weeks we had left in the project. By doing a root cause analysis, we identified a tangible problem we could fix. We pulled different people together and identified more than one potential cause of the problem. The rest of the team and the CEO supported us once we decided what we had to do.

#### The Five Why's and When to Stop



A challenge to using the Five-Why technique (similar to our root cause diagram) is knowing when to stop. If the cause is outside your sphere of influence, then stop. Strive to do something small that improves things, even if it isn't the perfect approach. Over time, the small things add up.

### How We Kept Focus During the Last Weeks

During the last couple of weeks of the project, the developers and managers worked closely together to fix any problems that could derail the final release. For example, the CEO made sure a senior developer was available to speed up fixes in core modules.

The team made the deadline. Although this had seemed like an unreachable goal just two months before, passing the finish line felt like any other day instead of an extraordinary achievement. The fact that the client signed on

for another project a week after release was the official acknowledgment of a job well done.

We all know that meeting a tight deadline can be done if you work a massive amount of overtime. But the team learned something else. Right after the final release, the project manager noted, “You know what? The team doesn’t do overtime anymore.” That was music to my ears. We had learned how to achieve more of the right things by doing less!

## What Lessons We Learned

No one single thing made this work, but a combination of small things reinforced each other. Kanban helped show the team where the problems existed (or didn’t exist). How we fixed the problems was up to us.

The information was cheap. All we had to do was put a Kanban board on the wall and start using it. Nothing else had to change. We kept the ticket tracking system, the project structure, and the teams. As soon as we identified a problem, we fixed it. We didn’t worry about trying to put in the best process as long as what we did was good enough.

Our implementation of Scrum was certainly less than perfect, but let’s look at what we didn’t do. We didn’t go back and order more of the same medicine that we were already taking. We didn’t mark ourselves against an ideal process method like “real Scrum” and start improvements from there. We couldn’t have—we didn’t have the trust capital to do such a thing, and we certainly didn’t have the time to make random bets if we wanted to save the project.

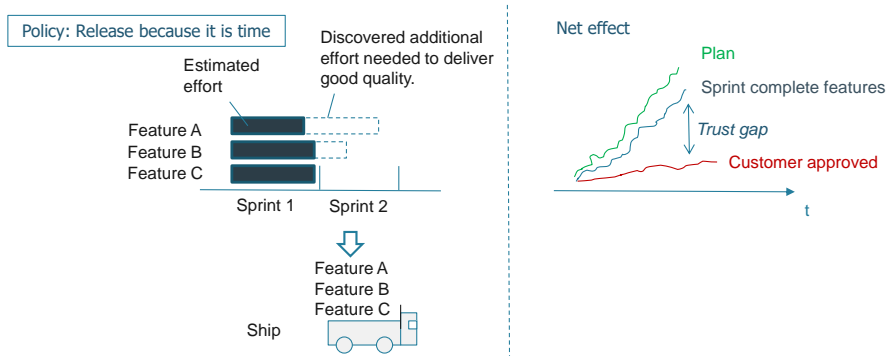
What did we do? We visualized where we had real flow problems and solved one problem at a time by focusing on quality. We wanted to get on that upward trajectory.

## Why Our Scrum Implementation Didn’t Work

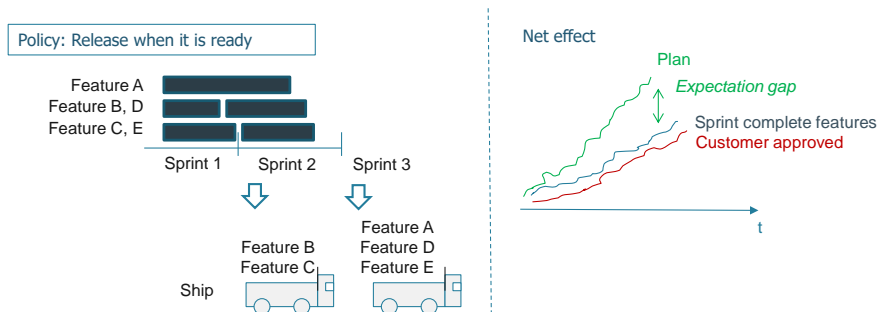
A few reasons account for why our Scrum implementation didn’t work out, including rework and wasted effort, not refactoring the code, and the lack of cooperation between teams. Let’s look at what we were doing wrong.

Features were designed to fit into sprints, but we didn’t take into account that we might not be able to finish development during that time frame. If we saw that we might have bitten off more than we could chew, we might have reduced scope so that we could at least finish parts of it, with plans to revisit the other parts at a later time. The next sprint focused on a different feature with a higher business value, and the cycle repeated. Because the partially finished feature was released, the plan showed progress, but it didn’t match

what was delivered. An invisible but growing trust gap formed between all parties involved, as illustrated in the following image.



When we moved to Kanban, we committed to working on the feature until it was finished and released. When the features got released often deviated from the project plan, but because this was communicated up front, trust wasn't lost (see the image that follows).



Another problem was that features were prioritized by business needs. Mounting pressure to get the project back on track meant that important refactoring decisions kept getting held back. Even if the developers could convince the project manager to let them refactor parts of the code, they wouldn't be able to fix everything within a single point. The sense was that refactoring was a futile effort.

Trying to refactor a key component built by an external vendor required cooperation from the team, platform specialists, managers, and the client. Even though the team was able to identify the problem, it couldn't be solved unless multiple stakeholders agreed it was a problem that needed fixing. We learned to involve multiple stakeholders to solve these problems instead of isolating everyone into distinct sprints.

Could we have tried other solutions? Yes, a given problem always has more than one answer. So what else could we have tried?

One option would've been to make our sprints longer. We didn't think the client would have agreed. I'm pretty sure that the message "We're going to work in longer sprints and keep away while we do" wouldn't have inspired a positive reaction.

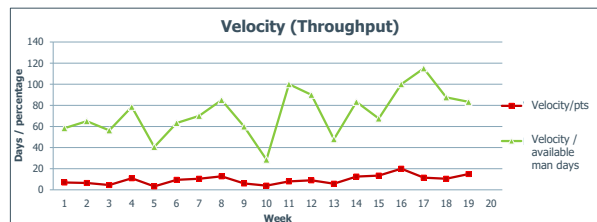
We could've changed our definition of *done*, except we weren't the only team involved. We had no control over the client's deployment process. If we changed the definition of what it meant to be done, the client was the one who had to deal with the consequences. We accomplished something similar by giving the client more visibility.

That was my view. Let's take a look at some metrics and hear what the people involved had to say.

## Comparing Now and Before

The team has since then implemented most of the testing frameworks. The reason I mention this is to show that you can overcome difficult technical hurdles and project management barriers. In the beginning, we couldn't envision ever having the time to put these fixes in place, but it happened. Many of the skills the team learned were transferred to the client's development team. The next step is to get the client's IT and project management teams on board.

We tracked our velocity throughout our project and can clearly see that we improved:



When we look at velocity in story points (the red line), we see that we averaged 7.25 story points per week in May, compared to 13.5 points per week in September. The velocity, normalized by the total number of man days (green line with the vertical axis denoting percentage), shows the May average was just 0.64, compared to September's 1.04.



I can't think of a better way of summing up than sharing the views of a team member.

The hardest part is to train yourself to embrace a new mentality.  
To realize that if a task is coded, this does not mean the task is completed.  
Kanban “forced” us to think about quality. When you have a column called  
“Function test” on the board, it’s a little bit hard to ignore it.  
It forced us to stop the coding, coding, coding chain.

I think the best lesson we have learned is to always think about quality.  
—Mariana, developer

## Make Your Own Improvements

It's easy to fall into the trap of making short-term decisions when you're under heavy pressure. And when things don't get done, it's easy to blame others and their shortcomings rather than figuring out what's stopping them from doing a good job.

Visualization is essential in knowing what needs to be addressed. Good leadership is important, too.

The first step in good leadership is to clarify the common goal.

Try to resist the temptation to postpone improvement actions and defer decisions. Procrastination will set you back right off the bat. A good leadership strategy is to always do something, regardless of how small, to improve the current state. The effect of improvements is cumulative, so don't underestimate the effect of small improvements. By making many small improvements, you're also setting a good leadership example; people will do what you do, not what you *say* you would do.

The second thing to do in good leadership is to follow up with vigilance on improvement actions taken. Nothing derails trust and breeds resentment more than people not pulling their weight. Visualizing the improvement action and asking participants to report their progress in front of everyone else makes them accountable. This is a good way to follow up on improvement actions taken.

Stick with one improvement at a time. Investing in quality is always a wise choice.

## Next Steps

The turnaround by the development teams showed me the importance of staying cool under pressure and investing in quality, even when the customer is screaming for something else. The urgency of the project meant that we

could not afford to put one foot wrong when we selected our improvements. Kanban helped with that precision. Kanban was also instrumental in helping the team and the customer to stay in focus, honoring commitments.

Let's leave the software realm for a bit and have a look at something completely different. How about Kanban outside IT? Can Kanban improve business operations? Check out the story of Kanban in the back office at a fast-growing bank.

# Using Kanban in the Back Office: Outside IT

---

When we talk about change, it's easy to get bogged down with a specific method or technique.

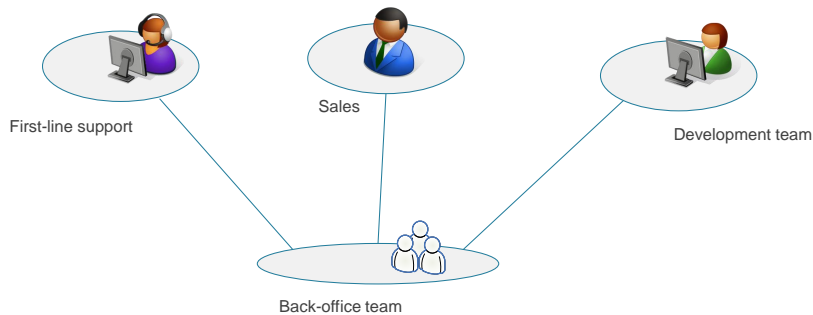
We've already discussed how Kanban can be applied to knowledge work. The next logical question is whether Kanban can work outside of IT. Can Kanban help a self-managing team in an enterprise office environment?

Let's take a look at how a company used Kanban to help a non-IT team improve its operations. Kanban helped this non-IT team to get an overview of its workload, prioritize the tasks, and make sure it was working on the right things.

## The Challenge: Keeping Up with Growth

In this case study, we look at a back-office team at a fast-growing bank in Sweden. The team handled pension issues and life-changing events for their customers, such as signing up new customers, processing payments and pension transfers, and tracking marriages and deaths. The team consisted of 14 people divided into two sub-teams—one with a consumer focus and the other with a corporate focus. Most of the time, the team interacted with a call center that acted as the bank's front-line support team. Occasionally, they fielded direct requests from sales and IT teams, as shown in the [image on page 76](#).

This non-IT team needed to keep up with the company's growth. The team needed to be able to add new members and get them up to speed quickly to keep up the pace instead of relying on a few senior members to handle the daily tasks.



The back-office team was already a very tight-knit team. We used Kanban to build on that relationship.

## How We Got Started

The team's Kanban journey started because the managers wanted a new way for the team to handle its workload. At first, the managers looked at lean for office work, but that didn't fit. The managers observed how IT used a Kanban board and decided to give it a try. At the time, I had no prior experience using Kanban outside of IT, but we decided it was worth a try.

Because team members were self-organizing, focused on their specific tasks, and expected to take end-to-end responsibility for their work, getting the full picture for the entire team was hard. This was a problem for management, which needed to track certain types of demand. Each demand type had explicit customer expectations associated with it, so it was important not to lose track of them. Some work had to be delivered on time to meet customer expectations and regulatory requirements.

Early on, we thought Kanban would help us get an overview and see what each member of the team was working on. Management also wanted help prioritizing tasks better to see whether the team was focusing on the right things. Other reasons for pushing forward with the change included spotting areas the team could improve, such as bottlenecks and stalled work.

We were concerned that Kanban would result in unnecessary overhead, but the positive effects—focus, basic structure, and getting an overview—outweighed those concerns. The team decided to give Kanban a try, and we soon had our first board up and running.

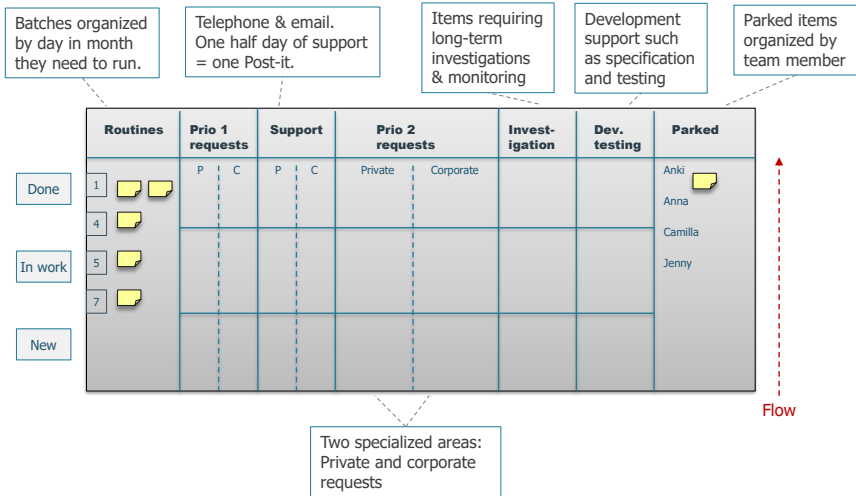
We invited the team to a workshop where they could experiment with Kanban, learn basic principles, and ask questions. For the most part, the team reacted positively. No office team in the company had tried using Kanban before, and this back-office team was excited to be viewed as pioneers.

### Learn the Principles



Learning about the principles behind Kanban helps the team understand how to move beyond the tools. Whenever you are about to introduce new teams to Kanban, invite all the members to a half-day workshop to make sure they understand both the what and the why of what they'll be doing. This will pay off in later conversations.

That's the background in a nutshell. Let's look at how the process worked.



## How Our Process Worked

Let's take a look at our Kanban board. The work arrived at any time through any number of channels: face-to-face requests from teams in the same building, queries over the phone, and tasks sent over email, to name a few. As soon as work arrived, it would be put on the Kanban board in the New row at the bottom and in the corresponding column, as shown in the following image. Each column has a designated prioritization, with the highest priority on the far left. The relative priority for each work type was set by the two managers of the teams. This made it very easy to spot what to focus on with a quick glance at the board from left to right.

Let's take a look at each of the columns in detail.

Demand Type	Column / Description
Routines	Some of the work was recurring. For example, "On Mondays we have to run the payment batch." All these routines were kept on the left side of the board in a column organized as a (monthly) calendar. When the day in the month arrived, the employee would fetch the routine Post-it from the Routines column and insert it as a normal item into ongoing work (Prio 2 Requests). After completion and calculation, it would be put back in the Routines column under the correct date.
Prio 1 Requests	These were requests from customers that needed to be delivered quickly and on time.
Support	Support was divided into blocks representing a half day's support work. So if a team member spent one morning on handling support, that would constitute one block and would be represented on the board as one Post-it. The reason we kept the support work on the board was to make it clear who was working on what, to get a rough idea of how much of the team's total workload came from support issues, and to learn the variations in the demand flow over time.
Prio 2 Requests	These were requests from internal functions that were not necessarily time critical.
Investigations	Long-term issues were kept here. Investigations represented issues we had to monitor or wait for input for a longer period of time, such as life-changing events when a customer died.
Development Support	This included acceptance testing, review of new requirements, or contributing ideas to the development team.
Parked	An item could only enter the Parked area if the team had already done all it could and external action was pending to resolve it. The neat thing about this column was that it was organized around each team member. This way, parked items didn't get lost or forgotten.

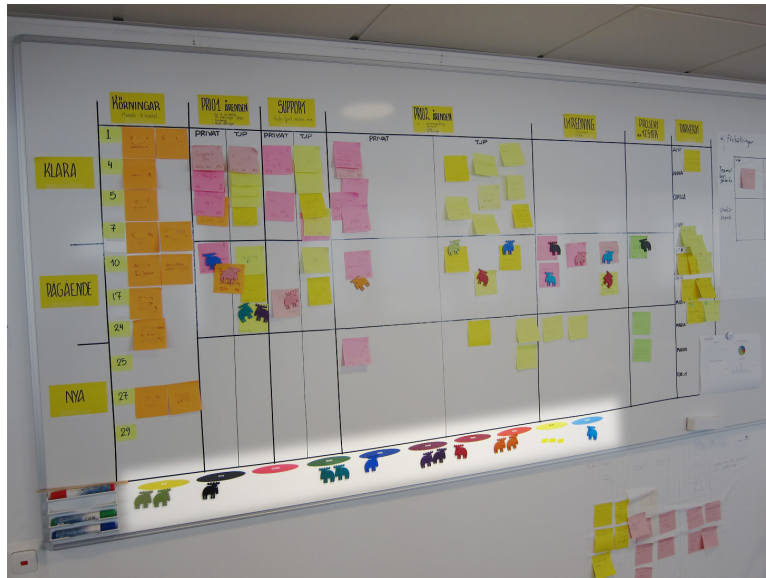
### Dealing with Parked Items

A challenge in maintaining a *parking area* for items that require external interaction is that it's easy to forget about them. You can use one of three approaches to avoid this:

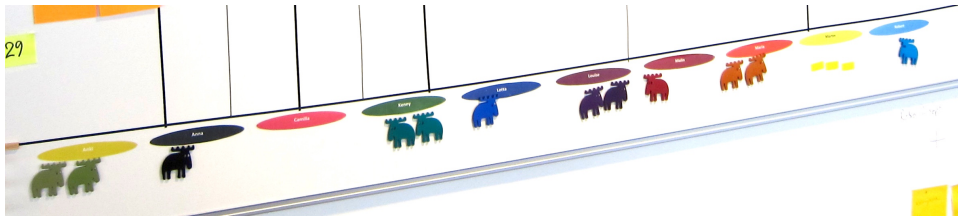


- Before you park an item, you should make sure that you've completed everything on your part first, because revisiting work can be costly.
- You should walk through all parked items in one of the weekly stand-up meetings.
- Or, as in the case of this team, you should keep parked tickets in a specific column organized by team member.

In the beginning, the team experimented with different settings and layouts to learn the level of granularity needed for each task, what to keep and what not to keep on the board, how to handle recurring routine work, and how to apply and use their work-in-progress (WIP) limits.



We introduced WIP limits with elks (yes, the animals, as you can see in the image [shown on page 80](#)). Every team member received three elks to place on the board. If the person had all the elks on the board, then they couldn't start new tasks until all the existing work had been completed.



Using a limited number of physical tokens helped everyone to see who was working on what and to track work at risk of stalling. Work can stall for various reasons, including people being away from the office or being sick.

Before this team started using Kanban, support requests and small questions would pour in to random persons on the team. To deal with this, we assigned the daily task of answering telephone calls and minor email requests to two team members. We rotated this assignment among the team members on a regular basis. Setting up this request-fielding system was challenging because not everyone on the team knew how to handle the main bulk of the questions coming in. After discussing the problem with the team, we decided that the person on duty would become the owner of that question. That person could ask for help from others, but as the owner, it was that person's responsibility to deliver an answer. Combining telephone and email support freed up other members of the team to work on long-term projects. The support schedule was also kept to the left of the Kanban board.

The members were positive about being part of the team. They liked the variety of work and often named good colleagues as one of the upsides of working with the team. With everyone managing their own workload, sharing responsibilities such as keeping the board updated and alerting each other when someone else was stalled didn't come naturally. We grew this sense of shared responsibility by making the team take ownership of the board. Everyone had a chance to weigh in on any change—we called them *team agreements*—before it was added to the board. We cultivated a strong sense of ownership by keeping a document on the left of the board listing all the agreed-upon changes (see the [image on page 81](#)).





The managers maintained a knowledge plan listing who needed to learn what topics and who could teach which materials. This plan was visible to the team, and it was up to the team to update and maintain the plan every six months.





## How We Continuously Improved

We figured out which areas to improve just by observing the board and regularly talking with the team at the daily stand-up meetings. The team held a retrospective meeting once a month in front of the board to find out what worked well and what needed to be improved. We visualized ongoing improvements under an *improvement Kanban*, shown in the following image, which we kept on the right of the main board.

	New	In progress [2]	Done
Team			
Org.		In progress [1]	

The improvement Kanban was divided into two swim lanes (horizontal rows), one for the improvements within the team's scope of control and one for improvements within the organization's scope. The department managers owned the Organization lane and were in charge of updating the team during the retrospective meetings.

The team measured total throughput, the distribution of demand types in percent, and cycle time (time through the Kanban system). Management updated the measurements once a week and posted them at the side of the board. The team found they successfully answered 95 percent of all requests within six days or less and addressed the majority of requests—at 88 percent—within two days.

Roughly every six months, the team revisited all the routines and asked questions such as, “Is this useful?” or “Should we do this differently?” Kanban forced the team to clarify how and why it was doing certain things. One of the managers told me, “Walking through our routines regularly has proved powerful. We’ll definitely keep doing this in the future.”

One of our early concerns was how to improve the software tools being used. The development team behind the software worked in the same building and used Scrum, but it was nearly impossible to get the back-office team’s feature requests prioritized because there were always other bigger requests with higher priority.

Bugs were fixed right away, though. So the team started inviting individual developers to sit next to the team member for an hour or two and observe how the software was being used. The development team then decided each developer should take his or her laptop along during this observation period. Whenever a member of the back-office team ran into difficulties with the software, the developer could fix and test the code on the spot.

That made a huge difference! The developers managed to fix an array of small but nagging issues in a short time—things such as flexibility when entering Social Security numbers in different ways and illogical button positions.

---

#### Get Out of the Sprint and Sit Next to Your User

---



There’s no better way to fix problems than to have the developer on the spot, sitting right next to the user.

---

How we approached making improvements with other teams may be new to you. We called it *fika-driven improvements*. *Fika* is a Swedish word meaning “coffee break,” or the time spent chatting over a cup of coffee. This team used *fika* to kick off improvements with the first-line-of-support team.

#### How We Got Going with Fika-Driven Improvements

*by: Manager*

We initiated our rolling support scheme with two team members handling smaller questions coming in over the phone and email. To make this work, we needed a new telephone number to allow for two incoming lines from corporate users and private clients. We also created a wiki with answers to frequently asked questions. We decided to share the wiki with first-line support so that they could answer some basic questions before calling us.

But we didn’t ask the first-line support to change their work routines right away. We hardly knew each other, despite working for the same company.

We started with a simple goal: to get to know the person behind each name. We invited members from first-line support over for coffee once a week. Our team got to know everyone in first-line support, and they got to know all of us during these coffee sessions. Once we had that relationship, we were able to discuss how we could work together to improve our work, such as with these issues:

- How we treat each other on the phone when under stress.
- How to handle duties that didn't logically belong to first-line support or back office.
- How to train a contact person in first line on answering questions that were a bit more advanced.
- How to create a small quiz that could be used to train new staff members in the first line to quickly get them up to speed on back-office issues.

The magic about this team was that they always managed to find a solution. Even when problems stretched outside their sphere of influence or when things seemed impossible, they took on problems with a positive spirit and managed to move forward every week. Some weeks they took smaller steps than others, but they always moved forward. The manager's role in driving this culture was critical. By asking questions and having conversations, the team always knew where they were and where they needed to go. A positive vibe around this team made everything seem possible.

---

#### Know the People Involved

---



Getting to know the people on the teams you want to involve in a change is a wise tactic.

---

We can learn plenty of minor tips and tricks from this team. But let's zoom out a bit for now and take a look at how the team felt about working with Kanban.

## What Lessons We Learned

When I asked the team's manager whether Kanban was useful, I got an interesting reply:

"When demand is high and we're under stress, the activity around the board increases and the team uses our Kanban board. But when the demand is low and if we only have two to three people at work, the board is considered an overhead since two to three people can easily have a shared overview on things and know they will complete work in time anyway. But when demand peaks, Kanban helps us prioritize and make sure that we focus on the right things. Prioritizing correctly would be nearly impossible otherwise."

One of the challenges in the beginning proved to be setting the right definitions to make Kanban useful in everyday life. For example, we had to agree beforehand on how routine work should be handled, whether it should be on the board, how to deal with small support tickets, and where to set our WIP limits. It took some experimentation to get those right. But once we settled on how to do it, Kanban worked just fine.

Another challenge we encountered was trusting people on the support team to handle questions coming in by email. The team used a common email list, so it was tempting for other members who knew the answers to step in and respond. This was something we constantly needed to talk about and work on, especially because this team was growing and we constantly had new people joining the team.

Let's look back at the questions we had before we got started and see how Kanban changed things.

## Comparing Now and Before

In the beginning, we asked whether Kanban could be applied to a normal office environment and under what situations Kanban would be valuable. Our early experience showed that Kanban helped the team get an overview of what was going on at any point in time. This allowed the team to move from working as individuals pulling tickets off queues to coming together as a team.

As the team members increased their skill and experience to handle a wider range of requests, the Kanban board was useful during high peaks of demand. During lower periods, the team's ability to self-manage combined with the support rotation was sufficient to handle most situations.

The low process overhead and self-managing stance allowed managers and senior team members to invest time in training and preparing new team members so that they could get up to speed and be self-sufficient quickly. Managers spent less time micromanaging and could shift focus to making improvements. A case in point: managers trained a new team, first-line support, to handle small routine requests.

In retrospect, Kanban was instrumental in helping the team grow and handle work efficiently during high peaks of demand as the company expanded.

## Make Your Own Improvements

Kanban helped the back-office team manage a growth phase. If you want to use Kanban in an office environment, here are a few things to bear in mind:

- Start with the team. Clarify and discuss your reasoning for introducing Kanban. Try it out for a period of time and then evaluate.
- Keep it simple. Use only a minimum of parts that are helpful to reduce unnecessary overhead. If you have the budget, consider an electronic board so you can get both support tickets and other work in the same place. An electronic board offers team members the flexibility to work from home, with the same overview as a physical board at work.
- If you face demand that has to be handled within a certain time frame but you lack the means to control the inflow (such as an emergency ward during a major natural disaster), make sure you can switch with other less time-sensitive work in a controlled fashion. In this back-office team's case, work at the far right of the board was swapped in favor of work at the far left during high peaks of demand. Kanban can help create a shared priority across all team members to make sure that you focus on the right things.
- If you move on to using an electronic Kanban board, try to combine it with a large touchscreen, where the team can gather around and discuss the issues. Kanban should be sparking conversations among team members about current work, blockers, upcoming demand, board layout, and improvements. A screen provides a focal point for sparking these conversations in the same way as a physical Kanban board would.

## Next Steps

The back-office team clearly demonstrated to me that Kanban works well indeed outside IT, especially in the case where the team is growing quickly.

If you're curious about how a Kanban team that has been using Kanban for more than 10 years is faring today, you'll want to read the story of Kanban in capital markets.

# Using Kanban in Capital Markets

---

*When I wrote the first edition of this book, I wanted to have a chapter that answered these questions: What happens with a Kanban team five years down the road? Do they still use it? What are they doing differently? And what happens if the same team tries out other process frameworks like Scrum and SAFe? Luckily, today I can share that story.*

Back in 2011, I met the Capital Markets DevOps teams. Throughout this chapter, I will simply refer to them as “the Capital Markets team.”

Capital markets is the division in a bank that trades in financial instruments—stock, bonds, and derivatives. The Capital Markets DevOps team handles the life cycle of the financial systems that makes that possible. The task is to run, maintain, and improve the trading systems and its connectivity for market data.

The bank where the Capital Markets team operated was active across Scandinavia. That meant that this team could not just take the perspective of what was happening in Sweden; they had to also consider the perspective from capital markets in the other Scandinavian countries, Finland, Denmark, and Norway.

When I first met the Capital Markets team, I found it interesting that they were seated on the trading floor, a stone’s throw from their users. A trader who needed help would walk over to the trading system team with the request. This also meant that when something wasn’t working, the team felt it immediately. The team ran DevOps colocated with their users.

## The Challenge: Solving the Seniority Catch-22

The Capital Markets team comprised of a group of 10 people, with a wide variation of expertise, experience, and age. Since the team had been colocated

next to their users, a habit had developed over the years that when the traders experienced a problem, they would walk over to the more experienced members of the team and ask them to fix it.

While this produced super-fast problem-solving, the flip side of the coin was a catch-22 situation: the easiest way to get a complex issue resolved was for a trader to connect directly with the most experienced team members, so freshly onboarded new team members never got a shot at those types of challenges. And if they did, their requestors would get impatient and ask the senior members to take over.

Over time, the compounding effect was that it was becoming harder and harder to grow the team.

Other unintended consequences appeared as well; technical debt was constantly deferred and stress was building up. Team members postponed taking time off, feeling a growing concern that the systems may not operate if they were not there to fix issues as they arose.

Daniel, who was the line manager at the time, together with a few pioneers in the Capital Markets team, decided that they wanted to address that. The first step on their journey was to introduce Kanban. Daniel was a key driving force behind the change and the journey that followed. Given that more than 10 years have elapsed since they started, I wanted to find out what happened.

What improvements took place along the way? What challenges did they have to deal with? To be honest, I had no idea if they had ditched Kanban altogether and moved on to something else. And if they switched to another process, what would that be? Let's hear their story!

Daniel is [shown on page 89](#) in front of the team's first Kanban board .

### **The Main Reasons We Got Started with Kanban**

*by: Daniel*

We got started with Kanban for three main reasons.

The first was the need to speed up improvement work. Important system consolidation work needed to be resolved, and one of those was a switch from local trading systems to a consolidated Nordic system environment. At the same time, overtime hours were mounting. Everyone was working flat out, but were we working on the right things?

The second reason was to enable a shift from a group of individuals to a well-functioning team. Before we started using Kanban, everyone had their own backlog, but there was little or no transparency to it. We wanted to bring transparency to the team's work, including the priorities, to the team and to outside the organization.



The third reason we made the shift to Kanban was to enable a shift in how we handled support. The direct one-on-one support where a user would walk over to a team member to get something fixed was from a user’s perspective very quick, but it also created huge bottlenecks; important long-term improvement work got interrupted and stalled, and unhealthy stress was piling up. We wanted to shift how “lights-on work” like support was handled to a more sustainable way that allows team members to go on vacation without feeling that everything depended on them —a shift that allows the team to balance long-term improvements with fast and nimble support.

## Don’t Wait for the Perfect Timing

Instead of waiting for the perfect opportunity to improve, the team decided to get started. This was for me a key lesson learned from working with them. There’s never going to be perfect timing. So you might as well start now.

The way they got started followed a journey similar to the other teams described earlier in the book. So we’ll make a small leap in time from how they got started to “what happened next?” Let’s take a look at the events that the team went through during their 13-year journey, from when they got started in 2011 to 2024.

The sequence of events is in chronological order to mirror the rough timeline in which they unfolded.

Name	Description
Split into two teams	<p>As soon as the team size went above 10, the team observed that not everyone was active at the stand-up meetings.</p> <p>The decision was taken to split the team into two units. They still were colocated and seated next to each other, and they also kept working with their shared Kanban board. The difference after the split was higher engagement from team members.</p>
Establish prioritization cadences with the larger organization	<p>The Capital Markets team established two prioritization cadences:</p> <p><i>Weekly business prioritization</i></p> <p>Every week, the head of Trading and the line managers of the Trading Systems team met and updated the prioritization of projects for the team.</p> <p><i>Monthly Nordic prioritization</i></p>



Every month, a forum was created to hash out the prioritizations from a Nordic perspective.

These cadences combined meant that the Trading Systems team had fresh input on three perspectives: the needs of the bank from a Nordic perspective, the needs of the Capital Markets department where they were working, and the needs driven by technical debt.

Team takes responsibility of trading systems availability	When capital markets open, the trading systems need to be up and running. Making that happen was achieved by heroic efforts from a few individuals. This shifted to the entire team taking over this responsibility. While this looked like a seemingly small thing from the outside, the cultural impact was big. It meant that the team can take responsibility for a complex machinery and making it work, lowering the stress and pressure on the individuals.
Structured support	Replacing the direct “go to your favorite person to get help” support pattern with a more structured one—one inbox that allowed the team to prioritize between requests and “pair up” to solve complex support items.

Name	Description
Know our capacity	<i>“When can you deliver X?”</i> To answer this question in a data-driven way (better than guessing), you need to know your capacity. By working with Kanban, the team could measure their throughput across different categories of work. That enabled the team and their line managers to give realistic forecasts on when items could be delivered.
Split users and developers into different floors	During the early years, the team was located next to their users (traders) on the same floor. The bank then decided to move all IT teams together, which meant putting users and developers on different floors. Slowly but steadily, users and developers drifted apart, with frequent misunderstandings and even conflicts emerging. Luckily, this was detected early on, and the Capital Markets team has moved back and is now located on the same floor as their users.
Scale to four teams	With a proven track record of operational performance, the Kanban team was now ready to expand, at first to three teams, and then to four. What is interesting is that the scaling happened without much drama or loss of performance.
Switch to a digital board	The final step was a switch from a physical Kanban board to a digital one. The benefit of the digital board was that the Kanban board was now available in real time, irrespective of distance. That enabled product owners, team members, and line managers to pull out the board in meetings, for example, to explain to stakeholders what they were focusing on, to revise priorities, and to insert new backlog items on the spot.
A detour to alternative process frameworks	See the following discussion.

## A Detour to Alternative Process Frameworks

During a period of 10-plus years, the organization where the Capital Markets team works has gone through a range of process frameworks. Given the long period of time that the Capital Markets team has existed, that was to be expected.

One anomaly in all those changes is that the organization never evaluated the performance of the Capital Markets team before or after each change—they just adapted. The problem with such a change approach is that the only way to demonstrate you’re doing what is asked is through compliance to the process.

So how did the Capital Markets team handle those changes?

## Run the Change as an Experiment

First, they weren’t afraid to try. Instead of resisting each change, they ran each change as an experiment to learn if the framework was helpful, and if so, which parts. Having this experimental mindset, not in words but in action, is commendable. It’s not something I often come across. It is generally the hallmark of a truly professional team.

## Experimenting with Scrum—Learnings from a DevOps Perspective

The team ran Scrum for a period of time. They found it was “too much planning for a DevOps team. Only a few hours after our sprint planning with the whole team, production incidents occurred, which threw most of the planning effort into the bin.” So while Scrum certainly can work well in development, for our DevOps environment, it was a poor fit.

## Experimenting with SAFe—Learnings from a DevOps Perspective

If you follow the development of the Capital Markets team, you’ll notice that they had already tackled two scaling challenges: first, growing the teams and second, addressing shared needs from other countries (sites). So in a sense, the Capital Markets team had already partly completed a scaling challenge.

The decision to run SAFe was a top-down request from the bank. Given that the Capital Market’s team weren’t newbies on either agile or scaling, what is interesting is how they approached making use of it.

**Thomas, chief product manager, had this to say:**

*Instead of running SAFe “out of the box,” we asked ourselves what components make sense to use in our context. We started to use those and dubbed our version “Punk SAFe.” That helped explain to the process compliance people and to our staff that our goal was not to follow the bible but rather to use the parts that were beneficial to our context.*

## Going Back to Kanban

Where are they now? After trying alternative frameworks, the Capital Markets team has now decided to go back to Kanban—at least one of the teams. If that experiment runs well, more teams will likely follow.

It's worth celebrating that the Capital Markets team was open to trying out alternative frameworks. They ran an experiment and evaluated it, and when improvements could be made, they pivoted.

After going full circle across different process frameworks over a period of 13 years, the Capital Markets team seems to have found their best fit to their context in Kanban.

## Lessons Learned

Once the team had a good idea of their capability and transparency in their workflow (using Kanban), they ran the organization's framework changes as experiments, only keeping the parts that worked. This allowed the team to both explore new ways and reduce waste. If you come to the point where you want to make the organization around you aware of wasteful practices and come aboard with a better approach, then you need data. Rightly operated, Kanban provides both data and transparency to operations. Both are keys to driving change.

The experimental mindset—the innovation engine—was Capital Markets leadership's contribution. It enabled continuous innovation and facilitated improvements in a high-pressure environment for more than 13 years.

## Make Your Own Improvements

Once you get started with a Kanban system, follow a similar recipe for your own improvement journey. Be transparent in how work is done, get data on your performance using flow metrics, and apply an experimental mindset to all changes. Run them as experiments, throw away the wasteful parts (it takes courage!), and keep the stuff that improves your team.

## Next Steps

In the final chapter, you'll find tips and tricks on going remote-first with Kanban, plus three important things to pay attention to when you take on change in your own organization.

# Using Kanban in Remote Work

---

In some situations you might be forced to work remote-first as a Kanban team. You might be geographically distributed, or weird office policies may prevent you from keeping a board. (It does happen—one team I know of had to hide their board in the emergency exit when the office police patrolled.)

And let's recognize that a digital Kanban board does have a few advantages.

One, for example, is real-time information. You can pull up the Kanban board at a meeting with a few mouse clicks if a stakeholder happens to ask you to take on a new request. By pulling up the board, and the backlog, you can justifiably make the call at the meeting whether to take on their request (if it's higher priority than what you have on the board) or say no or not now to it.

## Building Remote-First Kanban Teams That Work Well

The two tactics I recommend you follow are the change tactic and the tool tactic.

### Change Tactic: Develop the Team On-Site, Then Go Remote

When people meet in person, they build bonds of shared responsibilities. It matters because ultimately it's the people who make the work process improvements happen. Kanban is a tool to help visualize where improvements give the most bang for the buck. Making improvements happen often means many people need to move as one—as a team. So teamwork isn't just nice to have, it's an absolute necessity.

A few years back, I was able to interview one of the lead engineers of the open source framework Spring (Java Framework). Spring had been around since 2004, and like all frameworks, it had grown over time. All frameworks that

remain active that long easily get bloated, simply because of the need to maintain interoperability over time. The Spring modules, however, were able to maintain a remarkably good software architecture coherence. How did they do that?

I asked one of the lead engineers, “As an open source software framework, you are naturally distributed. What are the techniques you’ve used in Spring to keep a healthy architecture over time?”

He replied:

*“Two things: first, every module has a lead architect. The lead architect lays out the principles of architecture for their specific module. This means that while there will be variations of architecture between modules, we’ve managed to keep a healthy architecture over time in each module.*

*Second, we meet at least once a year. Yes, while most work happens remote-first (distributed across continents), if we didn’t meet face-to-face at least once a year, our remote-first operations would fall apart. Without meeting at least yearly at our conference, our remote-first operations would only be half as effective.”*

While modern remote tools have made huge leaps since then, the users of these tools are humans. The important thing to remember is that a bond of shared responsibilities develops when people meet.

When you start your Kanban journey, if you can, move people together during the first few days. During this time, create your Kanban board, set design principles, clarify definitions, get to know each other’s strengths and how to complement them. The investment will pay off. The team will find energy quicker, and they’ll go through the phases described earlier quicker.

Before pronouncing that this is *not* possible, carefully weigh the investment with the alternative costs and the extra time it will take to get the Kanban team to the same level of maturity.

### **Tool Tactic: Choose a Digital Kanban Tool Based on the Right Qualities**

The choice of tooling will impact the team’s ability to take responsibility for their process and to improve it. It can make or break your process. So avoid choosing a Kanban tool based on it being the default recommendation by the organization. It pays to carefully evaluate the qualities of a digital Kanban tool before picking it. It will save you time in the long run.

To help you with your tooling selection, here are three important qualities to look for:

### 1. A team should be able to change the board without having to ask for permission

Consider your Kanban board as almost being a live entity, a quiet shadowing team member; it needs nurturing and updates on a real-time basis. Changing the Kanban board should be as easy as hitting Settings and rearranging the board. Save. Done. Any threshold that blocks a team's ability to update their board on the fly means losing the real-time improvement benefit. For a physical board, updating it takes seconds; a digital one should be no different.

### 2. Status overview at a glance

Any team member working with a Kanban board should be able to see three things at a glance: What is my active task? What is my team's active task? and Is anything blocked? A Kanban team member typically flips through these status checks multiple times a day. This must be easy for a team member to do using simple keystrokes on their own computer.

We also have the team's perspective. Ideally, you want to have a large touch-screen that the team can huddle around and to show stakeholders what the team is up to. This means that the digital Kanban tool has to be capable of providing visualization of a readable version of the Kanban board on a big touchscreen as well as providing a good overview for a single individual on a local computer or smartphone.

### 3. Make good use of automation

This is where digital tools shine. The ability to create automation flows based on events and conditions can free up time by enabling batch operations. Here's one automation example: when feature is moved to Done, close all related stories and archive them.

Avoid hybrid meetings. The reason is simple: it's very hard for a remote participant to engage in a conversation if they only hear a part of it. Face-to-face conversations happen in duplex (two persons can talk at the same time without blocking each other), whereas digital conversations default to simplex (only one person can talk at a time).

It will be very tempting for members on-site to start discussing complex things face-to-face, which leaves remote participants out from the information flow. So keep the good habit of if one person is remote, everyone hooks up remote.

That said, don't be afraid to try out running Kanban remote. Experiment and improve in the same way as you would with a physical board.

## For Your Journey Ahead

This rounds off our case stories. I hope they inspired you with new ideas for making your own improvements.

When you take on change in your own organization, pay attention to these three important considerations:

- *Change takes courage.* As you read the stories, each improvement may look simple. But make no mistake, all of those changes took courage. The outcome was never certain. If you find yourself having second thoughts about your own improvement ideas, then ask yourself: *Has someone put forward a better idea? Is the change reversible?* If the answer to the first is a no and the latter a yes, it's time to make the leap.
- *The value of data.* One thing to pay attention to across all Kanban stories is that they all measured and visualized metrics—metrics that mirrored both output and operational performance. This value cannot be overestimated. Data is your friend. It helps build trust, and it helps provide a case when you're asking for help.
- *A Kanban team that doesn't measure metrics will have a hard time convincing anyone they are high-performing.* To be high-performing means paying attention to performance. It's such an easy thing to do when you have all the flow data in front of you on the Kanban board. Start capturing flow metrics on a regular basis; visualize it and use it both as flow improvement input and to celebrate improvements achieved.

In the Appendix, you'll find examples of Kanban boards that you can steal ideas from as well as tips and tricks on going remote-first with Kanban.

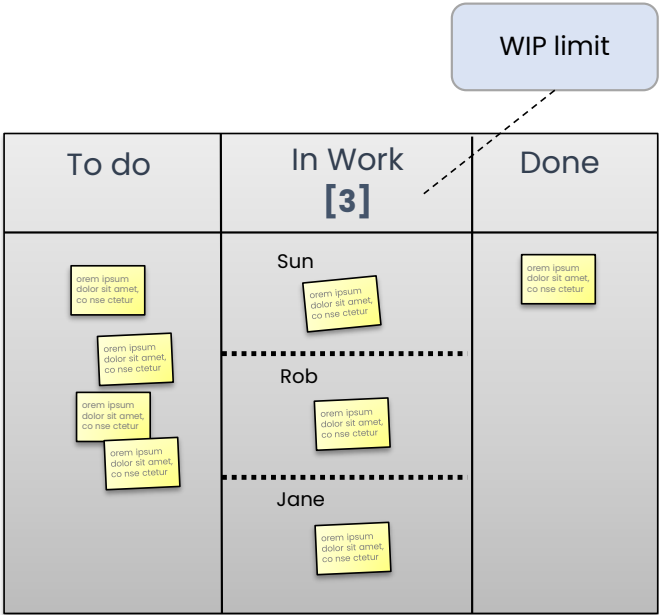
And now, I'd like to hand the baton over to you. What do you dream of improving at work today? What would unlock a creative flow of energy? Is Kanban a viable path?

If so, how can you get started tomorrow?



# Kanban Board Examples

## Product Development

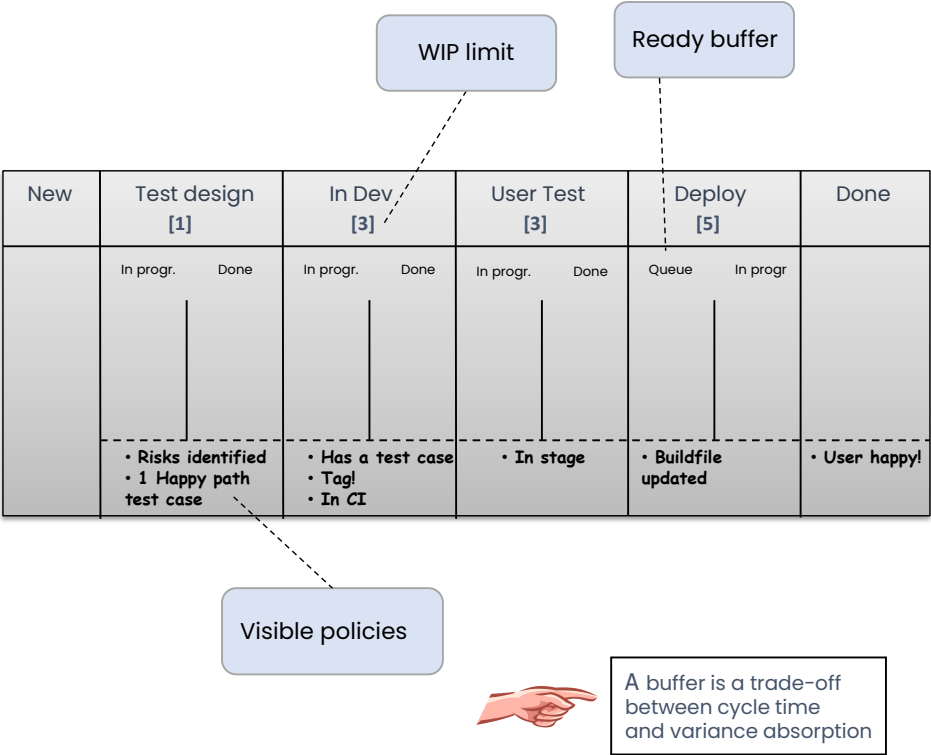


### Context



Scrum team applying WIP limits.

# Software Development Flow



## Context



Development teams responsible from Backlog to Live in production.

# Development Team with Multiple Clients

New	Estimate		Production issue				Done
			New	In Work			
<div><div></div><div></div><div></div></div>	In progr.	Done	Active projects				
			Test design [2]	Code [3]	Test [3]	Package [6]	

Classes of services in use:

Time constrained feature

Ordinary feature

Bug

Fixing tech debt

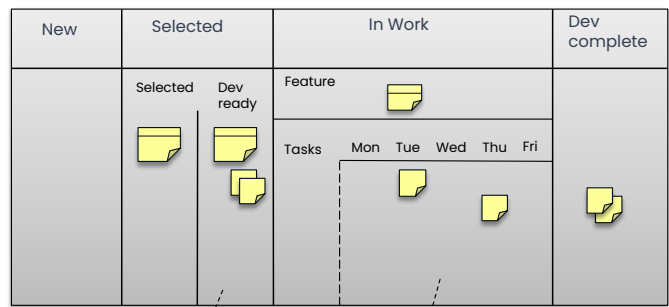
To risk balance your portfolio, limit the amount of each category allowed on the board at any one time

## Context



Software development teams that need to balance technical debt with customer needs, where the customer lacks the knowledge or incentive to manage the balance.

## Development Team Practicing Prediction



Credit: Chris Matts

If estimated duration is longer than 5 days, then task is broken down further

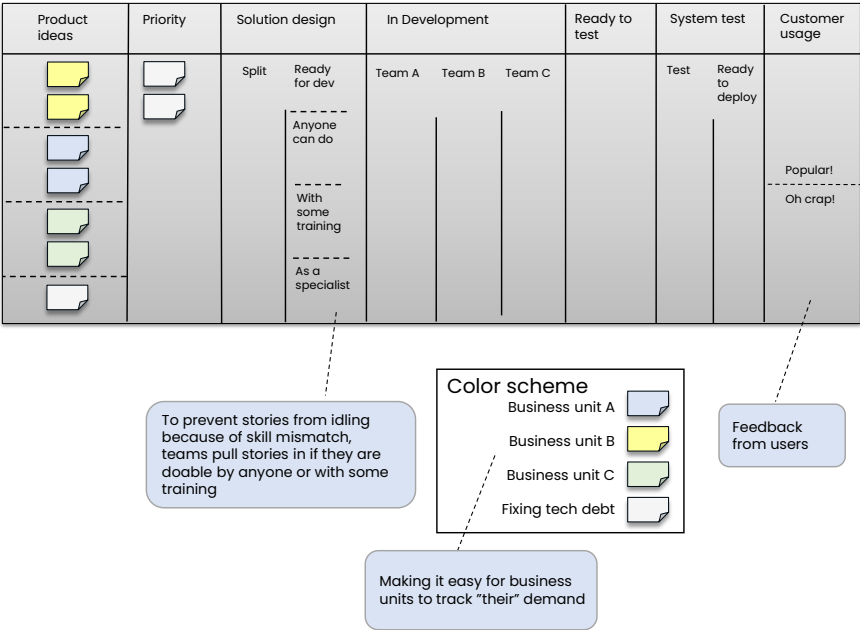
When developer starts a task, it is placed on the day they think the task will be done. The prediction is updated each day.

### Context



Development teams that practice improving prediction.

# Full Value Chain

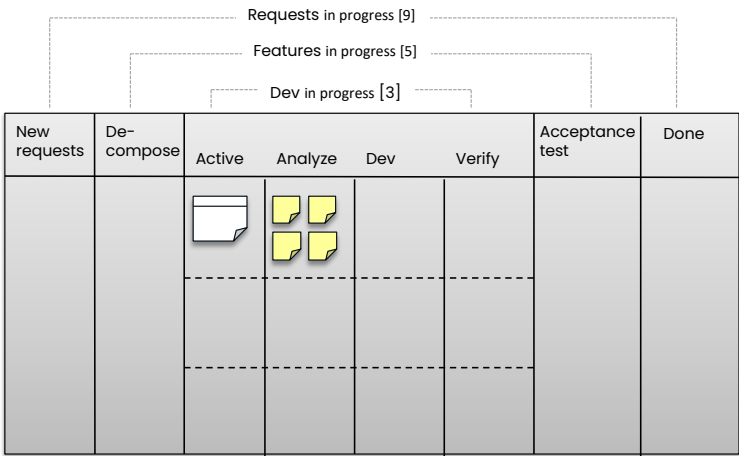


## Context



Full value chain—development teams serving three different business units.

## Mixing Product Discovery and Delivery



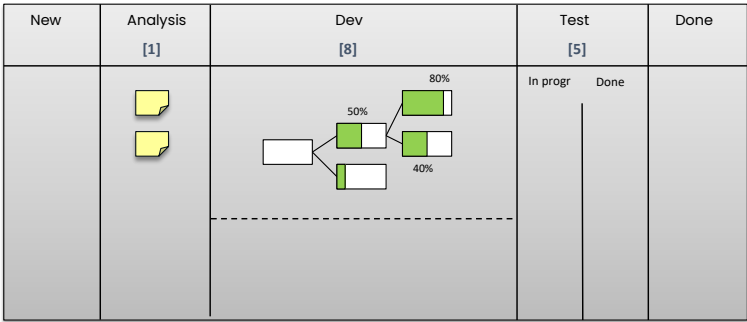
### Context



Allows business analysts, designers, developers, and testers to collaborate on product discovery and development, mixing the two. Useful during design-intensive phases of the project.

The Kanban board helps synchronize flow that's supported by both specialists and development teams.

# Progress by Architecture

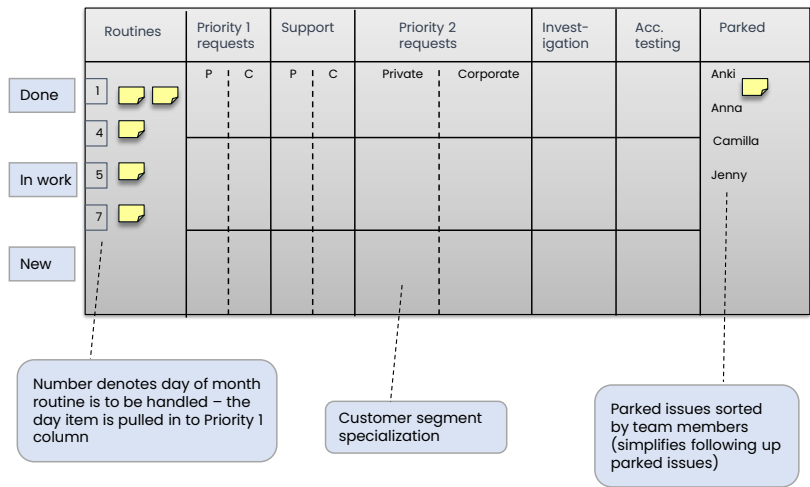


## Context



A hardware development team keeping track of progress through architecture. This makes status update on a complex project easily viewable at a glance.

## Business Operations



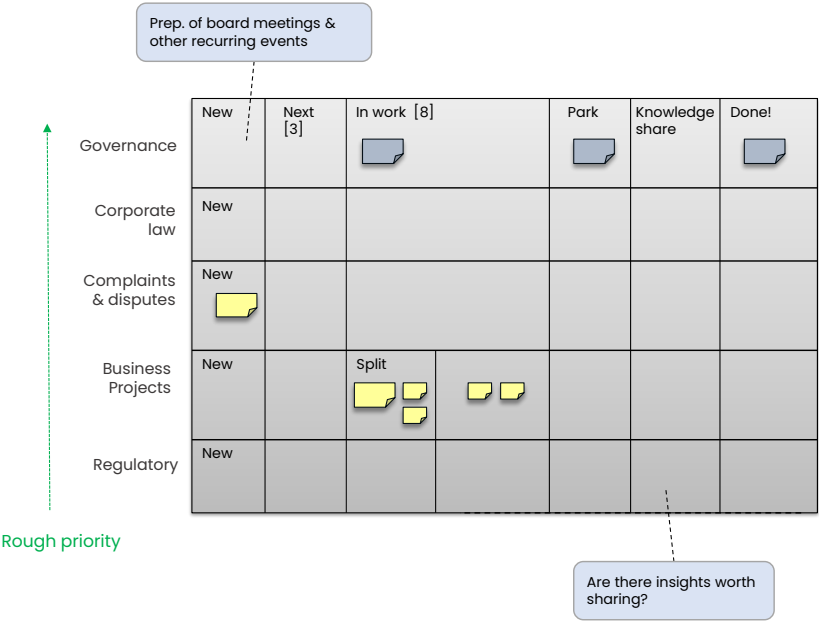
### Context



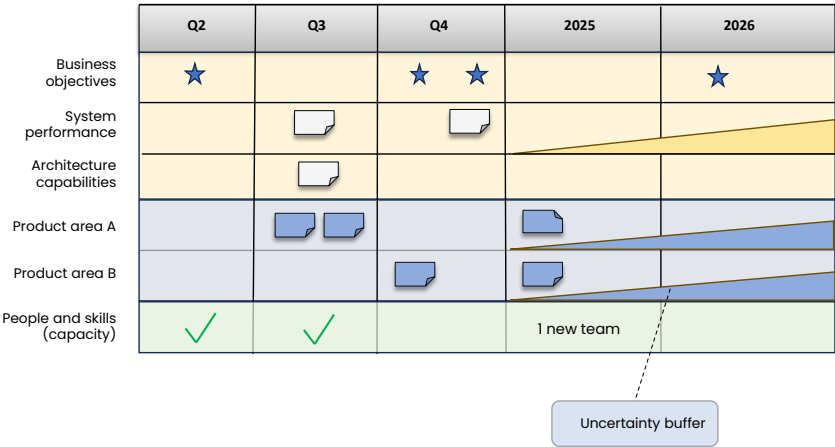
A business operations team solving customer requests.



# Corporate Legal



## Product Roadmap for Complex System (HW+SW)



Context




Product roadmap for complex system, for example, development of a self-driving car.

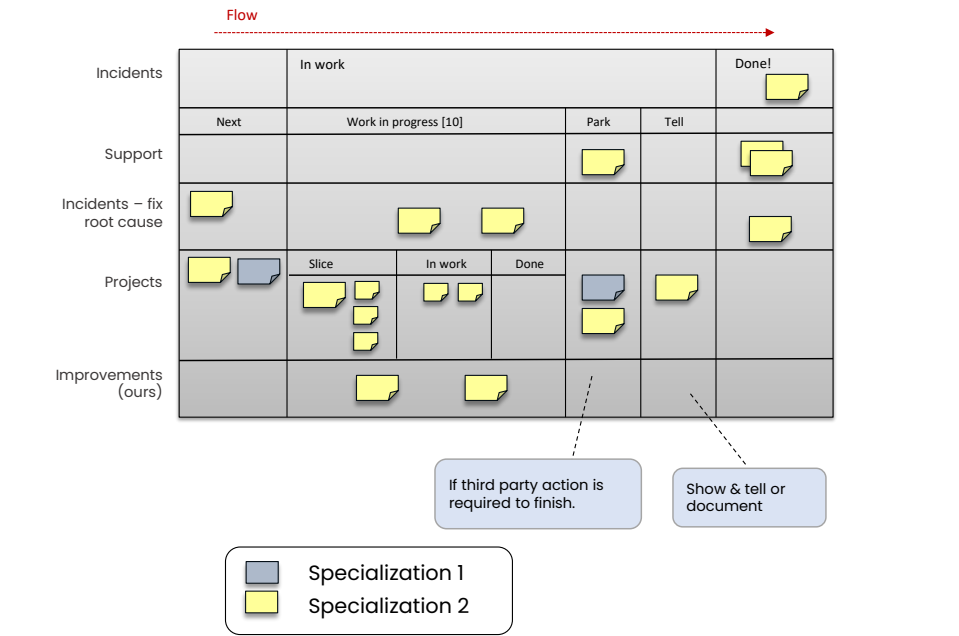
# System Administration Team




## Context

 A system administration team supporting multiple development teams and production sites. The Kanban board gives transparency to priorities, status, and what needs attention.

# DevOps for Online Platform



## Context



A DevOps team supporting production with high availability.

# Release Management

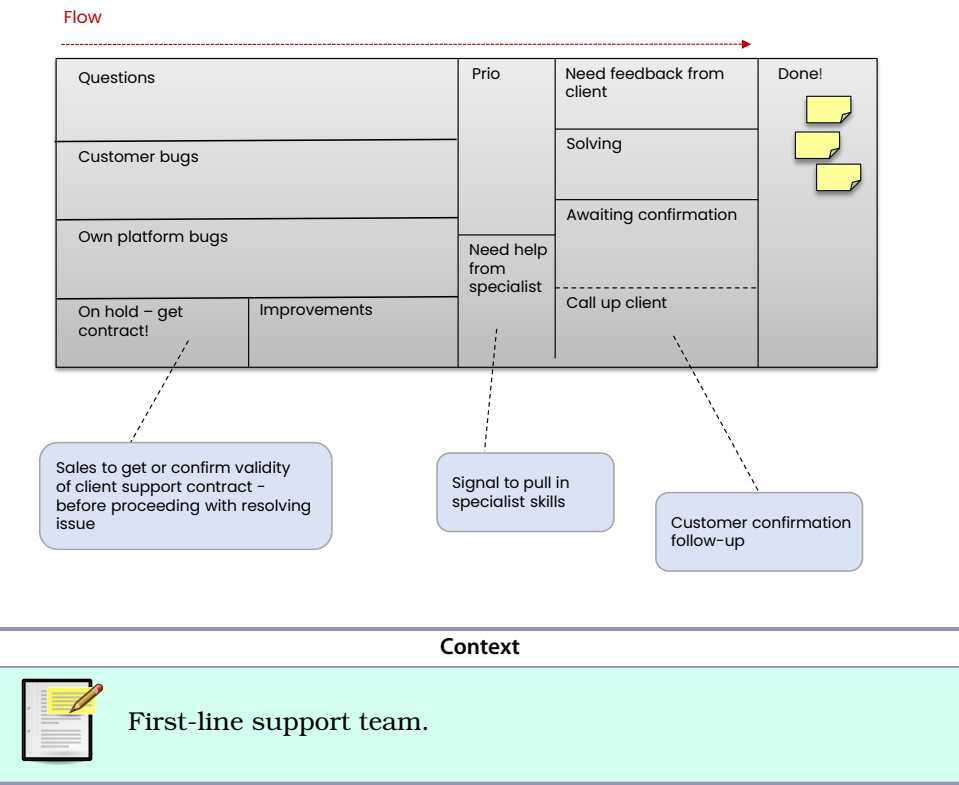
Incidents	In queue		In work				Done	
Release	Priority	Content	Prepare	Test			Go live	Follow up
	<div>1.3.1</div>	<div></div>		<div></div>	<div></div>	<div></div>		
	<div>1.3.2</div>							
	<div>1.3.3</div>							
Improvement	<div></div>		<div></div>					
Patch								
Problem								
Request								

## Context



A release management team managing software life cycle of 100 or more software products, stretching from in-house development to third-party products.

# First-Line Support



## Marketing and Sales—Sales Team from Lead to Deal

	Lead	Proposal Written	Under Negotiation	Won (verbal ok)	Purchase order received
John		Hot ----- Cold			
Alice		Hot ----- Cold			
Tintin		Hot ----- Cold			



"Key stuff is to make sure  
won (verbal ok) deals  
transforms fast into  
purchase order received"  
- CEO

---

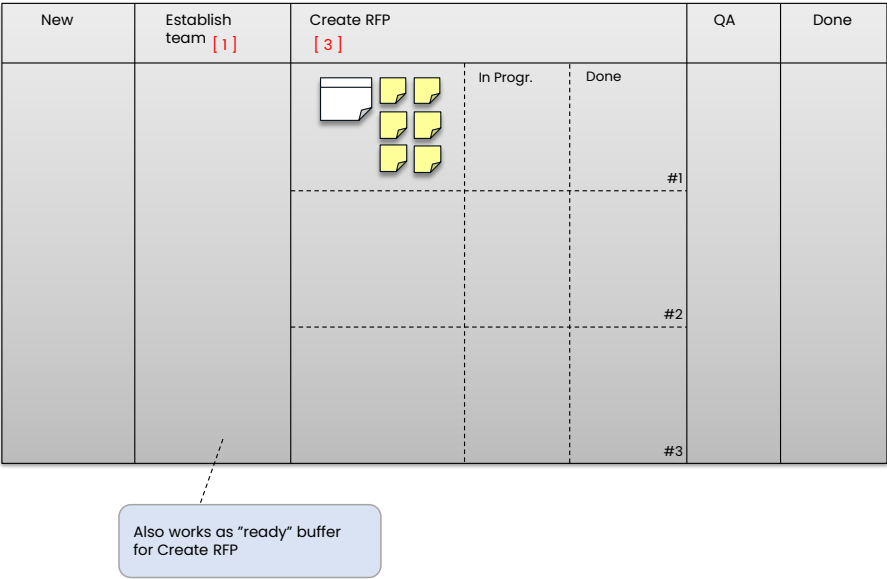
### Context



A sales team responding to managing the flow from lead to deal.

---

## Sales Team Respond to RFP






### Context



A sales team responding to more complex requests, customer RFPs (request for proposal). To do that, they first need to put together the right team to answer the request.



## Unpacking Vision and High-Level Goals

Vision	Defined (what could this mean for us?)	Validate objective value	Where do we want to be in			Done
			1m	6m	1y	
						
						

Enables team members to write down their point of view silently on a post-it first, then refine together

### Context



A management team that doesn't want to set and pass on fluffy company goals and vision statements. The Kanban board allows a team to quickly turn high-level visions and goals to practical applications and future aim. This acts as a small but valuable intermediate step, overriding the bad habit of simply passing on the fluffy goals to each management team member's function for goal breakdown. It also creates coherence and keeps silo mentality in check.

# Bibliography

- [And10] David J. Anderson. *Kanban*. Blue Hole Press, <http://www.e-junkie.com/129573>, 2010.
- [Bur14] Mike Burrows. *Kanban from the Inside*. Blue Hole Press, <http://www.e-junkie.com/129573>, 2014.
- [HS14] Marcus Hammarberg and Joakim Sunden. *Kanban in Action*. Manning Publications Co., Greenwich, CT, 2014.
- [Kni11] Henrik Kniberg. *Lean from the Trenches*. The Pragmatic Bookshelf, Dallas, TX, 2011.
- [KS09] Henrik Kniberg and Mattias Skarin. *Kanban and Scrum: Making the Most of Both*. InfoQueue, <http://www.infoq.com>, 2009.
- [Lik04] Jeffrey Liker. *The Toyota Way*. McGraw-Hill, Emeryville, CA, 2004.
- [MA12] Niklas Modig and Par Ahlstrom. *This is Lean*. Rheologica Publishing, Stockholm, Sweden, 2012.
- [PP03] Mary Poppendieck and Tom Poppendieck. *Lean Software Development: An Agile Toolkit for Software Development Managers*. Addison-Wesley, Boston, MA, 2003.
- [PP09] Mary Poppendieck and Tom Poppendieck. *Leading Lean Software Development*. Addison-Wesley Professional, Boston, MA, 2009.
- [Rei09] Donald G. Reinertsen. *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas Publishing, Redondo Beach, CA, 2009.

# Index

## A

A3 cards, *see* concepts, concepts  
accountability  
    transformation patterns  
        and, 8  
        trust and, 72  
active items only on Kanban  
boards, 44, 47, 55  
agile, *see also* lean; Scrum  
    advantages, 7  
    potential of, *xiii*  
    resources, *xiv*, 7  
Anderson, David J., *xiv*  
architecture Kanban board  
    example, progress by, 105  
archives, 97  
assignments, rotating, 50,  
    80, 82–84

## B

back-office case study  
    challenges, 75  
    improvement, 81–83  
    Kanban board, 76–81  
    lessons learned, 83  
    process, 77–80  
    rotating assignments,  
        50, 80, 82–84  
    support questions, 78  
blockers  
    change management case  
        study, 48–49  
    communication on, 21  
    Enterprise Kanban exam-  
        ple, 21  
    improvement pulses, 28

metrics, 29  
multiteam, 21  
stand-up meetings, 25  
visibility of and evaluat-  
ing Kanban boards, 48  
board owners  
    stand-up meetings, 24  
    team agreements, 80

bottlenecks  
    back-office case study, 76  
    change management case  
        study, 33–37  
    derailed project case  
        study, 61  
    Enterprise Kanban case  
        study, 33–40  
buffers, 100  
bugs  
    allotting time for, 24  
    back-office case study, 82

Burrows, Mike, *xiv*  
business operations, Kanban  
board example, 106

## C

capacity, capital markets case  
study, 89, 93  
capital markets case study  
    alternative frameworks,  
        91  
    challenges, 87–89  
    lessons learned, 93  
    timeline, 89–91

case studies, about, *xiii*, *see*  
    *also* back-office case study;  
    capital markets case study;  
    change management case  
    study; derailed project case  
    study; Enterprise Kanban  
    case study

celebrations, 55

change  
    adapting organizations to  
        frameworks, 8  
    communication on, 37  
    making small improve-  
        ments, 7, 42  
    overemphasis on individ-  
        ual vs. organizational  
        change, 8  
    principles of, 98  
    pushing vs. pulling, 6  
    random approach to, 8  
    soliciting feedback before,  
        13–14  
    surprises, avoiding, 37  
    transformation patterns,  
        7–9  
    transformation strategies,  
        42  
    trust and, 62

change management case  
study  
    bottlenecks, 33–37  
    challenges, 43, 50  
    improvement, 51–53  
    Kanban boards, 45–53  
    metrics, 51–53  
    process, 45–51  
    setup, 44–45  
    stand-up meetings, 49,  
        54

- team attitudes, 53
  - work flow, 48
  - changes, late, stopping, 34–37
  - coffee break–driven improvements, 82
  - collaboration
    - collaborative design meetings, 17–18
    - as core Kanban practice, 6
    - helping other teams, 55
    - product discovery, 104
  - collaborative design meetings, 17–18
  - color, organizing Kanban boards with, 46, 48
  - commits, testing and, 65
  - communication
    - on blockers, 21
    - Enterprise Kanban case study, 12, 40
    - on improvement initiatives, 42
    - on priorities, 55
    - shared language and, 14
    - soliciting feedback before change, 13–14
    - stress and, 83
    - surprises, avoiding, 37
  - company demos, 27
  - concept owners
    - adjusting concepts, 19, 30
    - collaborative design meetings, 18
    - stand-up meetings, 24
  - concepts
    - adjusting, 19, 30
    - canceling, 20
    - collaborative design meetings, 18
    - defined, 17
    - Enterprise Kanban example, 17–22
    - ownership of, 17
    - percentage of popular as metric, 28
    - size of, 19
  - confidence levels, polling, 61
  - continuous flow, *see* flow
  - continuous improvement, *see* improvement
  - costs, estimating, 22
  - courage, 98
  - creative height, 17
  - customers
    - customer use as end of process, 26
    - derailed projects and, 57, 62
    - feedback, 26, 40–41, 62
    - in Kanban board examples, 21, 26, 78, 101
    - observation by developers, 82
  - cutoff time, 35
  - cycle time, 81
- ## D
- 
- data
    - Kanban advantages, 93
    - value of in change, 98
  - decision-making
    - authority and stand-up meetings, 26
    - avoiding premade solutions, 19
    - based on documented processes vs. reality, 35, 66
    - clarifying decisions before addressing costs, 23
    - collaborative design meetings, 19
    - Enterprise case study, 22–24
    - pressure on, 72
    - resource-sharing decisions, 23
    - value uncertainty vs. cost uncertainty, 22
  - definition of done, *see* done, definition of
  - demand type, back-office case study, 81
  - demos, sprint vs. company, 27
  - derailed project case study
    - challenges, 57–59
    - improvement, 67–69
    - Kanban boards, 59–64, 67
    - lessons learned, 69–71
    - limiting work-in-progress (WIP), 59
    - metrics, 71
    - process, 64–66
    - sprints, 58–64, 69–71
    - testing, 61, 64–68
  - design meetings, collaborative, 17–18
  - developers
    - exchange programs, 66
    - preference for same developers in capital markets case study, 87–89
    - proximity to users, 82, 87, 89
  - development
    - in Kanban board examples, 20, 99–102
    - sprint demos and development bias, 27
  - DevOps Kanban board example, 110
  - discover phase, collaborative design meetings, 18
  - distribution of demand, back-office case study, 81
  - documentation
    - board changes, 80
    - collaborative design meetings, 18
    - decision-making based on, 35
    - ticket system as, 51
  - done, definition of
    - changing, 71
    - customer usage as, 26
    - derailed project case study, 61, 71
    - making explicit, 6
    - testing and, 61
- ## E
- 
- effort, estimating, 29
  - end-to-end flow, *see* flow
  - Enterprise Kanban case study
    - blockers, 21
    - bottlenecks, 33–40
    - challenges, 11–13
    - communication and, 12, 40
    - concept improvement, 30
    - improvement, 26–27, 37–42
    - Kanban board, 13–16, 18–30
    - lead time improvement, 29–42
    - lessons learned, 29–40
    - metrics, 27–32
    - organizational structure, 11
    - self-releasing, 35–37
    - setup, 13–16
    - stand-up meetings, 24–26

- stopping late changes, 34–37
    - team attitudes, 12, 40
    - workflow, 16–18
  - estimating
    - costs, 22
    - derailed project case study and, 58, 63
    - effort, 29
    - strategies for, 63
    - T-shirt sizing scheme, 63
    - time, 22, 30–32, 63
  - evaluation
    - frameworks, 92
    - Kanban boards, 48, 85
    - retrospective meetings, 51, 81
  - evolution
    - as core Kanban practice, 6
    - Kanban boards, 50
    - problem-solving and, 6
  - exchange programs, developer, 66
  - experimentation
    - evolution and, 6
    - experimental mindset, 92–93
    - making small improvements, 42
  - explore phase, collaborative design meetings, 18
- ## F
- 
- facilitators
    - collaborative design meetings, 17
    - stand-up meetings, 24
  - FAQs, 50, 82
  - features
    - Kanban board examples, 19, 23
    - prioritizing, 70
    - sprint times, 69–71
  - feedback
    - on boards, 26
    - before change, soliciting, 13
    - as core Kanban practice, 6
    - customer, 26, 40–41, 62
    - cycle time and, 62
    - in Kanban board examples, 21
    - thumb voting as, 13
    - validating usefulness of product, 27
  - fika-driven improvements, 82
  - Five-Why technique, 68
  - flow
    - defined, 3
    - focus on flow vs. sprints, 15
    - management of workflow as core Kanban practice, 5
    - product development flow, understanding, 2–5
    - transformation patterns and, 9
    - visualization of as core Kanban practice, 5, 42
    - walking during stand-up meetings, 25, 49
  - frameworks
    - alternative frameworks and capital markets case study, 91
    - evaluating, 92
    - fitting organization to, 8
  - full value chain Kanban board example, 103
- ## G
- 
- Go, ix
  - goals
    - clarifying, 72
    - high-level goals Kanban board example, 115
- ## H
- 
- Hammarberg, Marcus, xiv
  - hardware, progress by architecture Kanban board example, 105
- ## I
- 
- improvement
    - back-office study, 81–83
    - change management case study, 51–53
    - continuous feedback and, 27
    - as core Kanban practice, 6
    - derailed project case study, 67–69
    - Enterprise Kanban case study, 26–27, 37–42
    - evaluating, 35
    - experimentation and, 42
    - fika-driven approach, 82
    - improvement pulses, 28
    - making small improvements, 7, 42, 72
    - observation based, 35, 42
    - office environments, strategies for, 85
    - improvement Kanban board, 81
    - improvement pulses, 28
    - incident managers, stand-up meetings, 54
- ## K
- 
- Kanban, *see also* Kanban boards
    - advantages, 4, 98
    - core practices, 5
    - educating team, 77, 85
    - origins in manufacturing, 3–5
    - remote work and, 95–98
    - resources, xiv
    - rules in manufacturing, 4
  - Kanban (Anderson), xiv
  - Kanban and Scrum, xiv
  - Kanban boards
    - active items only, 44, 47, 55
    - back-office case study, 76–81
    - bottlenecks, identifying, 33–40
    - change management case study, 45–53
    - derailed project case study, 59–64, 67
    - digital versions, 15, 85, 89, 95–97
    - documenting changes to, 80
    - Enterprise Kanban case study, 13–16, 18–30
    - evaluating, 48, 85
    - evolution, 50
    - examples, xiv, 99–115
    - feedback on, 26
    - identifying bottlenecks, 61
    - location, 14–15
    - maintenance, 53
    - organization, 19–20, 23, 45–50, 53, 55
    - readability of, 48, 97
    - remote work, 95–98
    - setting definitions for organization, 84
  - Kanban cards, 3–5

*Kanban from the Inside*, [xiv](#)  
*Kanban in Action*, [xiv](#)  
 Kniberg, Henrik, [xiv](#)

## L

lead time  
   bottlenecks, analyzing for, [33–40](#)  
   change management case study, [51–53](#)  
   Enterprise Kanban case study, [29–42](#)  
   estimating, [22](#)  
   management of workflow as core Kanban practice and, [5](#)  
   metrics with, [28](#)  
 leadership  
   clarifying goals, [72](#)  
   follow-up and, [72](#)  
   levers of management, [1–3](#)  
   prioritization and, [55](#)  
   pushing vs. pulling, [6](#)  
*Leading Lean Software Development*, [ix](#)  
*Leading with Kanban*, [59](#)  
 lean  
   advantages, [7](#)  
   resources, [xiv](#)  
*Lean From the Trenches*, [xiv](#)  
*Lean Software Development*, [xiv](#)  
 legacy challenges, [11, 43](#)  
 legal team Kanban board example, [107](#)  
 Liker, Jeffrey, [xiv](#)  
 long-term effects, *see* capital markets case study

## M

magnets, [48, 79](#)  
 man-hours, [22](#)  
 management, levers of, [1–3](#)  
 manufacturing, Kanban in, [3–5](#)  
 marketing team Kanban board example, [113](#)  
 meaning, visibility of in evaluating Kanban boards, [49](#)  
 meetings, *see also* stand-up meetings  
   collaborative design meetings, [17–18](#)

hybrid meetings, [97](#)  
 remote work, [97](#)  
 metrics  
   back-office case study, [81](#)  
   blockers, [29](#)  
   change management case study, [51–53](#)  
   customer feedback as, [26, 40](#)  
   derailed project case study, [71](#)  
   Enterprise Kanban case study, [27–32](#)  
   improvement pulse, [28](#)  
   lead time, [28](#)  
   team performance and, [98](#)  
   time to market, [28, 30–32](#)  
   uses, [27](#)  
   visualizations, [29](#)  
 mixed product discovery and delivery Kanban board example, [104](#)  
 Modig, Niclas, [xiv](#)

## O

office uses, *see also* back-office case study  
   business operations Kanban board example, [106](#)  
   strategies for, [85](#)  
 overview, visibility of  
   back-office case study, [76, 84](#)  
   change management case study, [44, 53](#)  
   derailed project case study, [60](#)  
   digital boards, [97](#)  
   in evaluating Kanban boards, [48, 50](#)  
 ownership, *see also* board owners; concept owners; product owners  
   importance of, [16](#)  
   support questions, [80](#)

## P

parked items, [78–79, 106](#)  
 parking area, [78–79, 106](#)  
 policies  
   making explicit as core Kanban practice, [6](#)  
   vs. reality, [35, 66](#)  
 polling confidence levels, [61](#)  
 Poppendieck, Mary, [ix, xiv](#)

Poppendieck, Tom, [ix, xiv](#)  
 prediction, [102](#)  
*The Principles of Product Development Flow*, [xiv](#)  
 prioritization  
   back-office case study, [76, 78, 83–84](#)  
   by business needs, [70](#)  
   capital markets case study, [89](#)  
   change management case study, [45, 49](#)  
   communication and, [55](#)  
   demand and, [83–84](#)  
   denied requests, [49](#)  
   in Kanban board examples, [45, 49, 78, 83](#)  
   leadership and, [55](#)  
 problem-solving  
   change leadership and, [6](#)  
   collaborative design meetings, [19](#)  
   evolution and, [6](#)  
   Five-Why technique, [68](#)  
   identifying problems in derailed project case study, [57–59, 69](#)  
   information flow, [67](#)  
   root cause analysis, [67](#)  
   visualization, [72](#)  
 product idea success, [13](#)  
 product owners, [17–18, 98](#), *see also* concept owners  
 production  
   capital markets case study, [92](#)  
   Enterprise Kanban case study, [33, 36, 43–48, 51](#)  
   in Kanban board examples, [21, 101, 109](#)  
 products  
   product development Kanban board examples, [99, 104, 108](#)  
   product development flow, diagram, [2](#)  
   product development flow, understanding, [2–5](#)  
   product discovery, [104](#)  
   product roadmaps, [108](#)  
   validating usefulness of, [27](#)  
 progress  
   defining and sharing, [60](#)

polling confidence levels, 61  
 progress by architecture  
   Kanban board example, 105  
 pushing vs. pulling for change, 6

## Q

quality  
   derailed projects and, 62  
   focus on, 15, 41, 55, 69, 72

## R

regression errors, 65  
 Reinertsen, Donald, xiv  
 releases  
   change management case study, 45  
   checklists, 37  
   Kanban board organization, 48  
   release management  
     Kanban board example, 111  
   releasing when ready vs. on sprint schedule, 62, 70  
   schedules, 39, 62, 70  
   self-releasing, 35–37  
   stopping late changes, 34–37  
   testing on commits, 65  
 remote work, 95–98  
 request-fielding systems, 80  
 requests for proposals (RFPs), 114  
 resources  
   agile, xiv, 7  
   Kanban, xiv  
   lean, xiv  
 responsibility  
   accountability and trust, 72  
   capital markets case study, 89  
   stress and change management case study, 43  
   support questions, 80  
   visualizing with Kanban boards, 44  
 retrospective meetings  
   back-office case study, 81  
   change management case study, 51

effort metrics, 29  
 identifying challenges, 54  
 return on investment (ROI), 22  
 rework, 39, 62  
 RFPs (requests for proposals), 114  
 risk  
   decentralizing with concepts, 17  
   late changes and, 35, 37  
 ROI (return on investment), 22  
 root cause analysis, 67  
 rotating assignments, 50, 80, 82–84  
 routines, in Kanban board examples, 78, 82

## S

SAFe, 92  
 sales team Kanban board examples, 113–114  
 scaling  
   capital markets case study, 89, 92  
   transformation patterns and, 9  
 Scrum, *see also* sprints  
   back-office case study, 82  
   capital markets case study, 92  
   derailed project case study, 58–59, 69  
   Enterprise Kanban case study, 13, 17, 21, 27  
   product development  
     Kanban board example, 99  
 Skarin, Mattias, xiv  
 skills  
   developer exchange programs, 66  
   distribution of and rotating assignments, 50  
   knowledge plans, 80  
 software development Kanban board example, 100  
 solutions, avoiding predecided, 19  
 specialists  
   collaborative design meetings, 17  
   stand-up meetings, 24  
   waiting time and, 31

splitting teams, 89  
 Spring, 95  
 sprints  
   burndown, 58, 60  
   derailed project case study, 58–64, 69–71  
   progress, defining and sharing, 60  
   projects too complicated for single sprints, 62, 69  
   reintroducing, 16  
   shifting to continuous flow, 15  
   sprint demos vs. company demo, 27  
   timeframes, 62, 69–71  
 stand-up meetings  
   back-office case study, 81  
   change management case study, 49, 54  
   decision-making authority, 26  
   duration, 25, 49  
   Enterprise case study, 24–26  
   parked items, 79  
   participants, 24, 54  
   walking the flow, 25, 49  
 stress  
   change management case study, 43, 53–54  
   communication and, 83  
   root cause analysis, 68  
 Sunden, Joakim, xiv  
 support  
   back-office case study, 78, 80, 82, 84  
   capital markets case study, 87–89  
   change management case study, 50  
   FAQs, 50, 82  
   first-line support team  
     Kanban board example, 112  
   rotating assignments, 50, 80  
 surprises, avoiding, 37  
 system administration  
   Kanban board examples, 109  
   stand-up meetings, 54  
 system testing, 35, 39

---

**T**

T-shirt sizing scheme, 63  
 TDD (test-driven development), 64  
 team agreements, 80  
 teams  
   education on Kanban, 77, 85  
   metrics and team performance, 98  
   remote-first, 95  
   rotating assignments, 50, 80, 82–84  
   splitting, 89  
   team agreements, 80  
 technical debt  
   capital markets case study, 88–89  
   derailed project case study, 57  
   Enterprise Kanban case study, 20  
   multiple clients Kanban board example, 101  
 test-driven development (TDD), 64  
 testing  
   change management case study, 44–45  
   commits and, 65  
   definition of done and, 61  
   derailed project case study, 61, 64–68  
   Enterprise Kanban case study, 21, 34–35, 39  
   in Kanban board examples, 21, 61, 64  
   late changes and, 34  
   system, 35, 39  
   test-driven development (TDD), 64  
*This Is Lean*, xiv  
 throughput  
   back-office case study, 81  
   capital markets case study, 89  
   change management case study, 51–53  
   derailed project case study, 71  
   management of workflow as core Kanban practice and, 5  
 thumb voting, 13, 45  
 ticket system  
   change management case study, 44, 46, 48–50

derailed project case study, 69  
 documentation from, 51  
 Kanban boards with, 48–50

time, *see also* lead time  
   back-office case study and cycle time, 81  
   change management case study, 45  
   cutoff time, 35  
   derailed project case study, 63  
   Enterprise Kanban case study, 23, 30–32  
   estimating, 22, 30–32, 63  
   finding spare, 63  
   sprints, 62, 69–71  
   system testing, 39  
   time to market, 28, 30–32  
   value of, 1, 6  
   waiting time, 31, 35, 39

time to market, 28, 30–32  
 tools, reasons to use, 54  
 Toyota, 3–5

*The Toyota Way*, xiv

transcribing, collaborative design meetings, 18

transformation  
   patterns, 7–9  
   strategies for, 42

transparency  
   capital markets case study, 88, 93  
   evaluating Kanban boards, 48  
   product development and, 6  
   reviewing board and, 23  
   transformation patterns and, 9

trust  
   accountability, 72  
   change and, 62  
   derailed project case study, 62, 65  
   quality and, 62  
   releasing when ready vs. on sprint schedule, 70  
   rotating assignments and, 84  
   trust gap and sprint timeframes, 70

---

**U**

upper control limits (UCL), 30–32  
 usefulness  
   evaluating Kanban boards, 49  
   validating usefulness of product, 27

---

**V**

value  
   to customer in Enterprise Kanban case study, 40–41  
   decision-making and value uncertainty, 22  
   estimating, 22  
   full value chain Kanban board example, 103  
   value-adding time in Enterprise Kanban case study, 38  
   value efficiency and transformation patterns, 9  
   visualizing value stream, 42

velocity, *see* throughput  
 vision Kanban board example, 115

visualization  
   as core Kanban practice, 5  
   identifying patterns and problems, 53, 72  
   Kanban advantages, 44  
   ticket system and, 44  
   value stream, 42

vocabulary, shared, 14

voting  
   on challenges, 45  
   thumb voting, 13, 45

---

**W**

waiting time  
   estimates and, 31  
   testing, 35, 39  
 walking parked items, 79  
 walking the flow, 25, 49  
 work-in-progress (WIP), limiting  
   back-office case study, 79  
   as Kanban core practice, 5



derailed project case  
study, [59](#)

product development  
    Kanban board example,  
    [99](#)  
workflow, *see* flow

## Thank you!

---

We hope you enjoyed this book and that you're already thinking about what you want to learn next. To help make that decision easier, we're offering you this gift.

Head on over to <https://pragprog.com> right now, and use the coupon code BUYANOTHER2025 to save 30% on your next ebook. Offer is void where prohibited or restricted. This offer does not apply to any edition of *The Pragmatic Programmer* ebook.

And if you'd like to share your own expertise with the world, why not propose a writing idea to us? After all, many of our best authors started off as our readers, just like you. With up to a 50% royalty, world-class editorial services, and a name you trust, there's nothing to lose. Visit <https://pragprog.com/become-an-author/> today to learn more and to get started.

Thank you for your continued support. We hope to hear from you again soon!

The Pragmatic Bookshelf

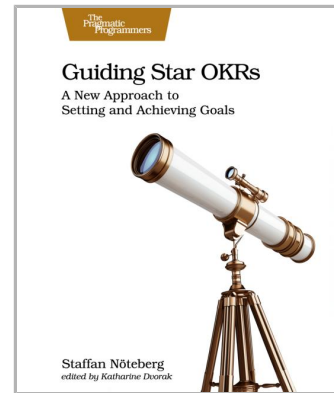


## Guiding Star OKRs

---

Tired of traditional OKRs that stifle innovation and demotivate teams? The Guiding Star OKR framework offers a refreshing new approach to goal setting, emphasizing purpose, unified direction, and adaptability. Best-selling author Staffan Nöteberg distills knowledge from diverse industries, teaching you to create a compelling “Guiding Star” vision that inspires, aligns, and empowers teams. Learn to foster intrinsic motivation, embrace continuous adaptation, and unlock strategic agility for sustainable success in today’s ever-changing business world.

Staffan Nöteberg  
(176 pages) ISBN: 9798888651285. \$42.95  
<https://pragprog.com/book/snokrs>

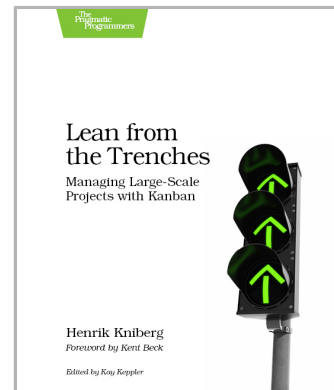


## Lean from the Trenches

---

You know the Agile and Lean development buzzwords, you’ve read the books. But when systems need a serious overhaul, you need to see how it works in real life, with real situations and people. *Lean from the Trenches* is all about actual practice. Every key point is illustrated with a photo or diagram, and anecdotes bring you inside the project as you discover why and how one organization modernized its workplace in record time.

Henrik Kniberg  
(178 pages) ISBN: 9781934356852. \$30  
<https://pragprog.com/book/hklean>



# The Agile Samurai

Here are three simple truths about software development:

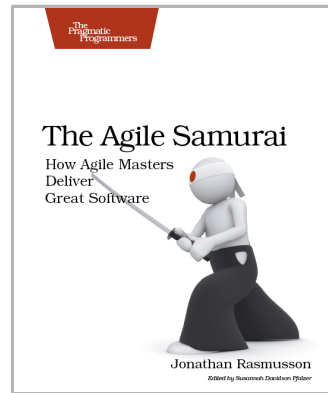
1. You can't gather all the requirements up front.
2. The requirements you do gather will change.
3. There is always more to do than time and money will allow

Those are the facts of life. But you can deal with those facts (and more) by becoming a fierce software-delivery professional, capable of dispatching the most dire of software projects and the toughest delivery schedules with ease and grace.

Jonathan Rasmusson

(264 pages) ISBN: 9781934356586. \$34.95

<https://pragprog.com/book/jtrap>



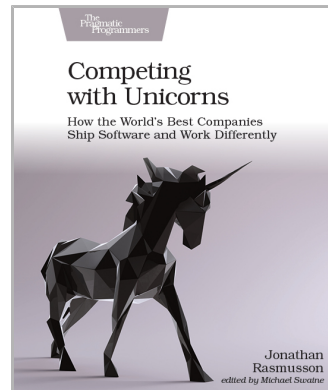
# Competing with Unicorns

Today's tech unicorns develop software differently. They've developed a way of working that lets them scale like an enterprise while working like a startup. These techniques can be learned. This book takes you behind the scenes and shows you how companies like Google, Facebook, and Spotify do it. Leverage their insights, so your teams can work better together, ship higher-quality product faster, innovate more quickly, and compete with the unicorns.

Jonathan Rasmusson

(138 pages) ISBN: 9781680507232. \$26.95

<https://pragprog.com/book/jragile>



## Agile Web Development with Rails 8

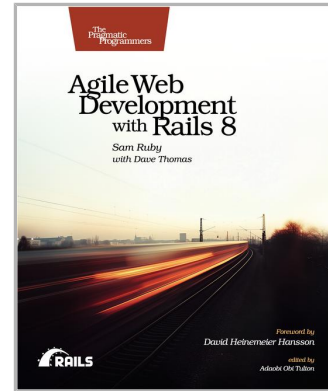
---

The eighth major release of Rails focuses on the ability to produce production-ready applications. It achieves this while building upon and retaining the ability to produce fantastic user experiences, and achieves all the benefits of single-page applications at a fraction of the complexity. Rails 8 introduces Kamal 2, Thruster, new database adapters, replaces the asset pipeline, and adds a new authentication generator. The result is a toolkit so powerful that it allows a single individual to create modern applications upon which they can build a competitive business—the way it used to be.

Sam Ruby

(488 pages) ISBN: 9798888651346. \$67.95

<https://pragprog.com/book/rails8>



## Next-Level A/B Testing

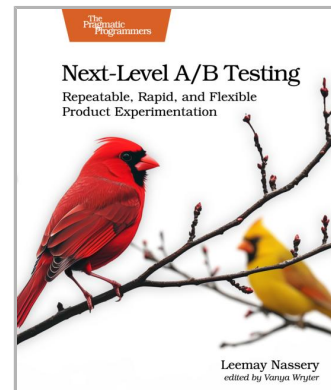
---

The better the tools you have in your experimentation toolkit, the better off teams will be shipping and evaluating new features on a product. Learn how to create robust A/B testing strategies that evolve with your product and engineering needs. See how to run experiments quickly, efficiently, and at less cost with the overarching goal of improving your product experience and your company's bottom line.

Leemay Nassery

(226 pages) ISBN: 9798888651308. \$53.95

<https://pragprog.com/book/abtestprac>



# The Pragmatic Bookshelf

---

The Pragmatic Bookshelf features books written by professional developers for professional developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

## Visit Us Online

---

### **This Book's Home Page**

<https://pragprog.com/book/mskanban2>

Source code from this book, errata, and other resources. Come give us feedback, too!

### **Keep Up-to-Date**

<https://pragprog.com>

Join our announcement mailing list (low volume) or follow us on Twitter @pragprog for new titles, sales, coupons, hot tips, and more.

### **New and Noteworthy**

<https://pragprog.com/news>

Check out the latest Pragmatic developments, new titles, and other offerings.

## Buy the Book

---

If you liked this ebook, perhaps you'd like to have a paper copy of the book. Paperbacks are available from your local independent bookstore and wherever fine books are sold.

## Contact Us

---

Online Orders: <https://pragprog.com/catalog>

Customer Service: [support@pragprog.com](mailto:support@pragprog.com)

International Rights: [translations@pragprog.com](mailto:translations@pragprog.com)

Academic Use: [academic@pragprog.com](mailto:academic@pragprog.com)

Write for Us: <http://write-for-us.pragprog.com>