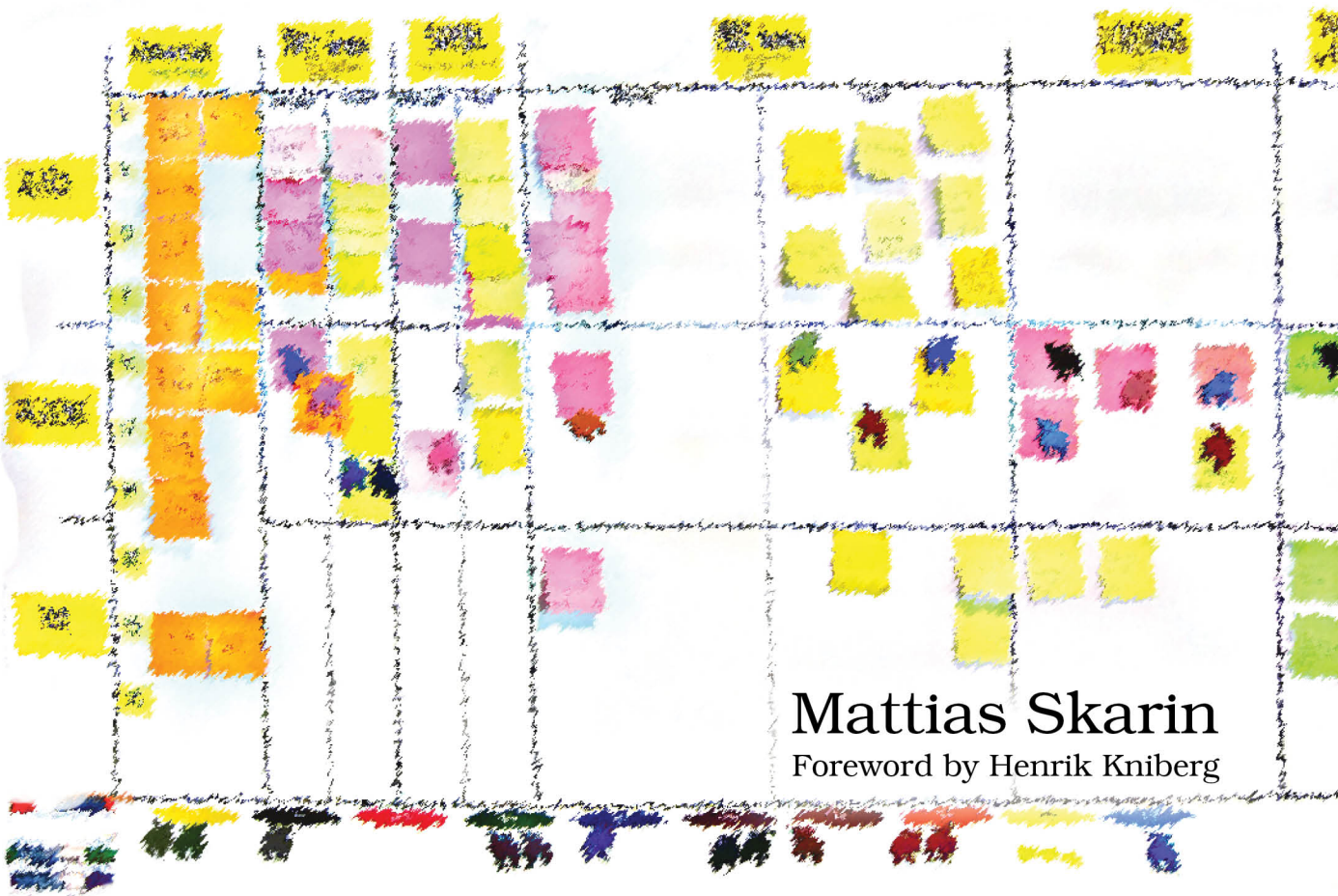
The Pragmatic Programmers

Real-World Kanban

Do Less, Accomplish More with Lean Thinking



Mattias Skarin

Foreword by Henrik Kniberg

edited by Fahmida Y. Rashid

Early praise for *Real-World Kanban*

This is a very practical and to-the-point book on how to implement the Kanban method. The four 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.”

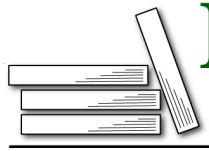
Real-World Kanban

Do Less, Accomplish More with Lean Thinking

Mattias Skarin

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *https://pragprog.com*.

The team that produced this book includes:

Fahmida Y. Rashid (editor)
Potomac Indexing, LLC (indexer)
Cathleen Small (copyeditor)
Dave Thomas (typesetter)
Janet Furlow (producer)
Ellie Callahan (support)

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-077-6
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—June 2015

Contents

Part II — Appendix

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 four 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.

Four 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 four 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*

Take Charge and Make Changes

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

In all four cases, the companies were well established, and each company carried legacy—entrenched systems, processes, organization, and mindset. All were under heavy competitive pressure and needed to find ways to simultaneously work smarter and deliver.

These companies didn't meet the challenge with short-term cost reduction overhauls or with another reorganization. Instead, they radically improved how the teams worked, making it easier for people to deliver products with great quality.

We followed a straightforward recipe for improvement in all four case studies: unlock the power of the initiative, transfer process ownership and responsibility of quality to the people doing the work, focus management on improving end-to-end flow (not just one portion), and grow an experimental culture where "let's try it out" is the automatic answer to every new idea.

All four cases tell the story of teams and leaders working in big organizations who decided to take charge of their future and succeeded.

Inside This Book, You Will Learn…

The four case studies illustrate what *really* happened when we used Kanban. You will see how teams:

- Improved time to market and cross-department collaboration across the full value chain using Enterprise Kanban on page 23.
- Improved teamwork and flow in change management and operations on page 57.
- Saved a derailing project on page 69. (And the team met the deadline!)
- Helped a non-IT team keep up with the company's growth on page 85.

And let's be honest, the improvements didn't just happen by putting a visual board on the wall. They happened because managers and engineers took charge of their situation. Sure Kanban can help us visualize what problems to solve. But it is doing something about them that makes the difference.

And that's important: no real improvements can be made without good leadership. In the *You Hold the Key to Your Future* chapter, I share the long-term thinking that helped guide our improvement efforts. This helped us keep the momentum going after the initial implementation and keep pushing for new improvement opportunities.

How the Book Is Organized

If you're already familiar with Kanban, Lean, and the underlying theory, feel free to jump right into the case studies. Each one has a different context, so start with the one that appeals to you the most. If you want to understand the thinking behind how we coached leaders, the *You Hold the Key to Your Future* chapter is the best place to start.

Each case study outlines a challenge and the journey we took to overcome it. The chapters walk through the challenge, how we got started with Kanban, how our process worked, and the lessons we learned. You'll see how we figured out whether things really changed for the better. Each chapter ends with a few nuggets of wisdom that you can apply if you're in a similar situation.

I set the stage for the case study in each chapter, and I pull out helpful tips and suggestions for you to follow. They will be marked with an *i* icon or a light bulb.

If you aren't familiar with the requirements tool *concepts*, check out the Appendix for more details. We used this basic tool to ensure that senior engineers were working with quality input. Concepts also helped us preserve the integrity of original ideas through all stages of development.

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 is 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]*

This book also has its own web page.[1] 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.

Acknowledgements

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 and Judith for their contributions in shaping and polishing each story.

What are you waiting for? Let's get started!

1. https://pragprog.com/book/mskanban/real-world-kanban

Part I

Lean Applied:
Four Stories of Improving Using Kanban

CHAPTER 1

You Hold the Key to Your Future

Magic happens when people are in the zone, when they can focus their creative energies on making a difference in the world. Positive things happen when they realize they can challenge and improve things that don't work, even when the problem spans across the organization.

This book tells the stories of Kanban implementations in four product-development scenarios from different parts of the value chain. None of the teams involved started from a picture-perfect position. All of them carried legacy systems and processes and had to work within a bigger ecosystem. What they did have in common was a will to make a difference. My message is this: if they can do it, so can you! You hold the key to your future. The actions you take every day shape your future.

Making changes can be easy or hard. It depends on your outlook and approach. You need courage to make change. More than that, you need to have many conversations with *everyone* involved. Sometimes, all you need is just time and patience.

The true art of improvement is all about making the most of opportunities as they present themselves. It's about grasping, exploiting, and leveraging opportunities to your benefit and advantage instead of staying put with the status quo. Management has a big role to play here. By building the right mix of culture, organization, and architecture, you can reduce friction and dampers on initiatives. I call this *leading using long-term thinking*.

Each case study in this book tells a story of how we found a way to improve time to market, collaboration, and focus. That doesn't mean that it was the only way, just what worked for us. If you just want to read the stories or borrow some ideas from the case studies, skip right to the next chapter, *Enterprise Kanban: Improve the Full Value Chain*. If you'd like to grasp a little

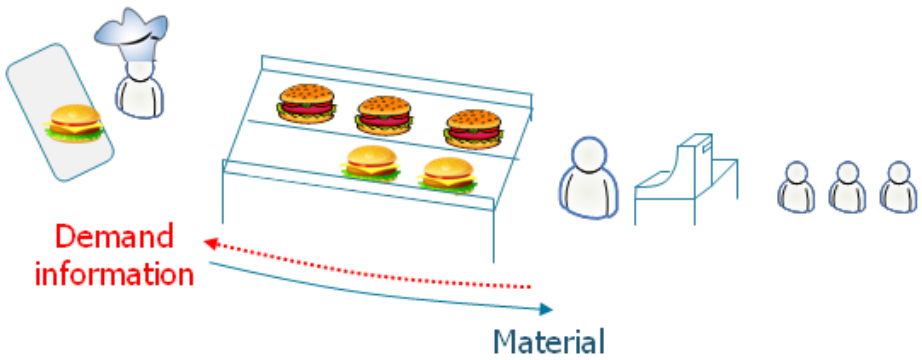
bit of the underlying theory and what we've learned from all of the case studies, read on. You will hear about how we recognized improvement opportunities and made decisions.

What Is 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 there is a need for parts. It is 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.)

What's the fuss about?

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 are done.

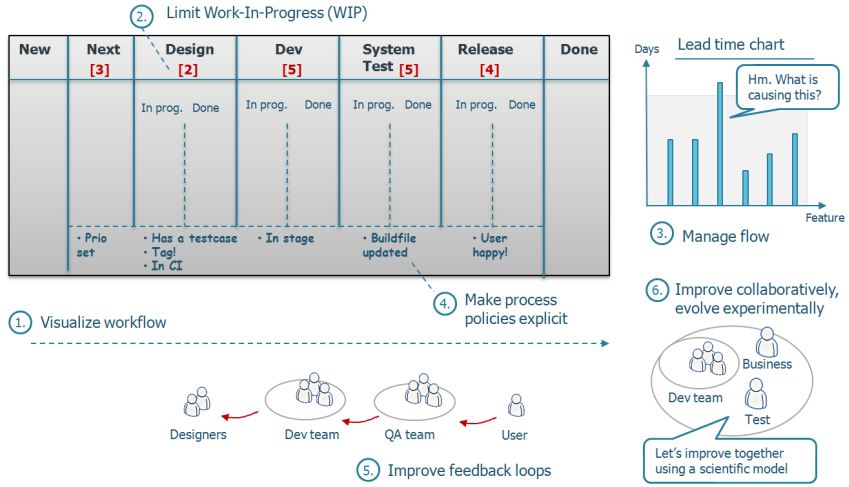
The Kanban Rules

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.

Now that we have a bit of historical perspective on how Kanban was originally used in manufacturing, turn the next page to 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.

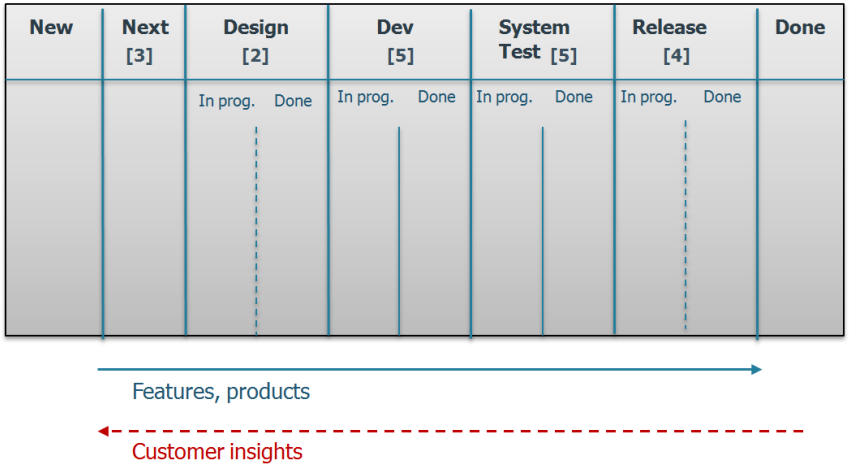
3. *Manage flow*. To improve, we manage our constraints and measure flow. The two most common measurements for flow are throughput and lead time.

4. *Make process policies explicit*. It is hard to make improvements if every team member has a different standard. An explicit policy is necessary so that there is a shared agreement among team members working with the Kanban board. An example can be 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 will need to implement a feedback loop as well to help you learn if you are 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 manufacturing can we apply to product development?



When we apply Kanban in product development, the emphasis tends to be put on improving the flow through product development (the blue arrow in the preceding figure) at the expense of improving the flow of information (ideas and customer insights, the red arrow in the figure). But the latter tells you whether you are on the right track.

Here is a rule: no defects can be passed on to the next stage. Passing known defects down the line is unconscionable—not only does it increase costs because of rework, it also wears down trust between people and teams. Trust is hard currency you can use when trying to get other departments to cooperate with changes you are trying to make. Don't lose trust.

Kanban is great in helping you visualize the current situation, but it doesn't tell you what to do. You need to decide that! This is not trivial. There are often limited resources to spend on improvements, and you need to make them count to keep up a positive momentum. To get some guidance on what to improve on, we need to expand our view and look at other methods, such as Lean.

What Is Lean?

Lean is a production practice that considers the expenditure of resources for any goal other than the creation of value for the end customer to be wasteful, and thus a target for elimination. That's a mouthful. What does that mean?

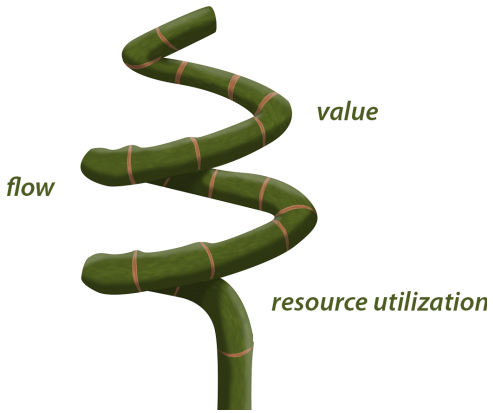
In Lean, we judge value from the customer's perspective. Why does this matter? It is all too easy for an organization to celebrate the success of

structure, such as process methods, organizational setup, and cost reduction, and to overlook the system output—lead time, quality, and customer experience. By putting ourselves in our customers' shoes, we can start to see the difference between doing and creating things of value.

But improving by eliminating waste is not a very effective approach in product development. A far more functional approach is to shift management's attention from optimizing resource usage efficiency (keeping people and equipment busy) to optimizing flow efficiency (time to market and throughput).

The final leap is to shift focus from optimizing flow to optimizing value. This means organizations go through three phases as they improve. They start off by focusing on resource usage efficiency, and then move on to flow efficiency, and finally to optimizing value.

The trick to pulling off each leap is that the organization has to give up or forgo something, or else it stays at its local optimum.



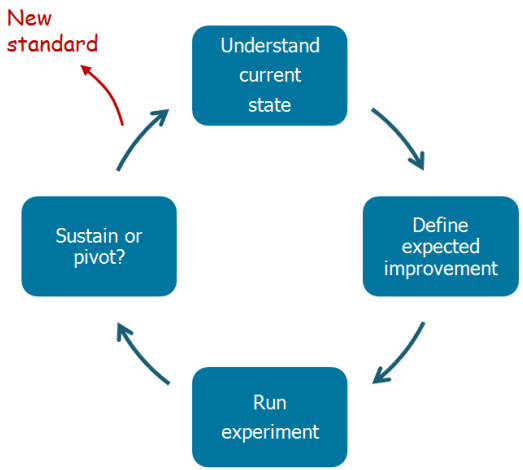
This is a challenge regardless of what process you use. Agile is no exception. It helps to keep in mind that our objective is not to implement the perfect process, but to make a difference in the world and to perpetually improve our ability to do so.

While passion is the driving force behind producing great things, creative and passionate people need to be supported by a conducive system that makes it easy for them to create value and fix quality problems. That is where Lean comes in. It's essentially a management system that continuously strives to take the annoying parts out of the equation so that people can focus and deliver real value.

That brings us to continuous improvement. The trademark of a Lean organization is its focus on continuously improving. To always be better *is* the

operational strategy. Compare this with a never-ending stream of cost-cutting and excellence programs initiated by each new CEO. The secret is to replace programs like these with something far simpler: transfer the ownership of quality and process to the people doing the work, and engage management in simplifying the effort needed to deliver one unit of value. When we improve how work is done, the resulting efficiency leads to a reduction in total costs. This is in stark contrast to letting cost-cutting measures dictate how the work gets done and having to cut corners and compromise on quality. The long-term benefit is that this strategy builds trust between people and management.

Improvement initiatives need focus and direction, especially in a collaborative setting. To ensure focus and to avoid political deadlocks, it helps to use scientific thinking and small experiments to improve. The good news is that this can be applied both to organizational changes and to changes in process and product. Let's walk through each element:



Understand current state. This describes where you are now. Understanding your current situation is the precondition for setting out a direction.

Define expected improvement. This describes what you expect to happen. By clarifying your hypothesis before running your experiment, both you and your peers stand a better chance of detecting unexpected outcomes (new information).

Run experiment. Try new things! In product development, we deal with interdependent factors and multiple solution options. Without further ado, run one or more experiments to find the best path to your target condition.

Sustain or pivot. If no new information has been generated during the timeframe as set out, you should consider killing the experiment and moving on. If it has produced a positive outcome, implement it as a new standard. A standard in product development can be a consensus on quality as work moves across organizational compartments, or it can be the skills needed to do quality work.

This is known as *PDCA* (Plan Do Check Act) in traditional Lean literature.

What can we learn from PDCA? While product development means moving through a fluid and uncertain territory, we can make great use of experiments and scientific thinking. Taking a fact-based approach to decision making helps preserve energy. The key to decisiveness while applying a fact-based approach in product development is to recognize that the goal is not to be *exactly right*, but to seek just enough information so that we can eliminate the unlikely options.

Find Opportunities to Improve

It's all too easy to get emotionally attached to your current approach.

The first lesson we learned was to avoid scoring ourselves against a process model, an organizational structure, or a tool. Instead, you should build a culture of curiosity and experimental thinking. Train managers to hone their insights by observing outcome and to suggest improvements from there. This will help you learn from new information even if it is at odds with your current understanding, and it will help you avoid getting too emotionally attached to your current approach.

While this book talks about process, what you won't see if you go looking for process artifacts is the amount of effort we put in each case to develop thinking people who take initiatives, managers included. The key to sustained progress is not so much about implementing a process, but about developing a conducive culture for thinking people to help you evolve—people who care about *what* and *why*. Managers who fail to nurture thinking people as their key objective will just copy and paste others' solutions. These managers tend to be oblivious when their current approach fails to work.

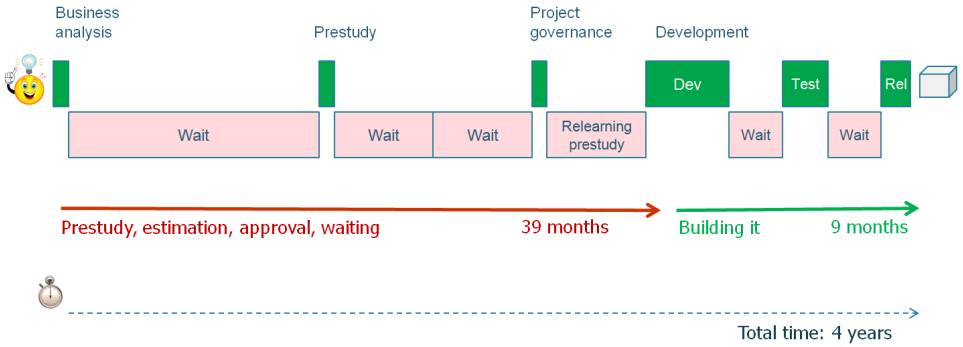
Finally, we learned to improve end to end, not just our little part. Having great people is not enough, because they could be trapped in a dysfunctional system where they are unable to ship any high-quality products. To see these challenges, management needs to think about optimizing the whole, not just the parts. Shifting their attention lets them see how ideas get diluted through

handovers and lets them grasp the delayed effect of pushing poor-quality products down the line.

A Change Should Never Come as a Surprise

A sanity check for long-term leadership is that a change should never come as a surprise. Follow this rule, and it will help you build up trust capital. This trust capital is hard currency when it comes to taking risk in periods of uncertainty.

Let's take a look at an example—a value-stream map over an idea's journey through a real company.



From this perspective, it's easy to spot improvement opportunities. In this case, the front-end part of the value stream (prestudy, estimation, approval, and waiting – 39 months) accounts for 81 percent of the total lead time. Even in Agile companies, the front-end parts of the value stream are generally quite intact and take up roughly 60 percent or more of the total lead time. The real trouble happens if this is spent planning rather than generating new information—for example, through market and user feedback.

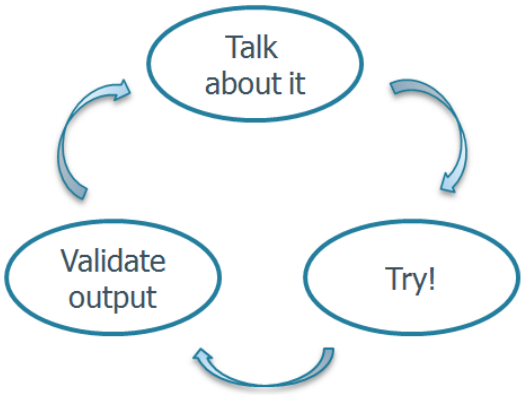
The problem is, we rarely see things from this perspective. The fact that the lead time is long would most likely go unnoticed among the people working on the project, because they have their nose so close to the grindstone that they see only their own part.

Another reason you don't see many people shifting perspectives has to do with our ingrained mental habits. When observing the world, our mind has to decide what information to act on. So if you see something that your mind doesn't think is important, you subconsciously ignore it. This is known as *change blindness*. So in a sense, we are predisposed to ignore and discard new information, even when it is staring us in the face.

The crucial first step in unlocking these improvement opportunities is to shift management's perspective from perfecting each part to improving the performance of the whole (for example, end-to-end lead time). The goal should not be to implement the perfect process, but simply to make it easy for great people to deliver value and to fix quality problems as soon as possible.

How We Got Buy-in to Change

Since it is our job to build thinking people who will be the ones to tweak processes and tools long after we are gone, we have to bear that in mind when we introduce changes. The way we introduce changes is actually really simple:



Talk about it. Talking to people before initiating a change is treating people you work with as thinking, intelligent beings. This is a necessary prerequisite to taking responsibility for the process and the quality produced. Talking with people gives them the chance to weigh in on what is going on and to provide better options.

Your leadership is only as strong as the conversation you are ready to have.

Try! People often have many opinions about how things should work. But there's only one way to find out how things really work, and that's by trying them out. It is often helpful to implement a change on an experimental basis for a limited time and then evaluate whether things get better.

If you meet resistance, start with the pioneers in the company. You don't need to wait for everyone to get on board. Seed curiosity by handing out books or visiting others who have made the journey.

Evaluate output. It's easy to confuse actions (fulfillment of your implementation plan) with real improvement. You only really know whether your actions generated the desired effect by observing real outcome (product quality, lead time, predictability on delivery, and so on). The first role to connect with output is management. As a rule of thumb, management is responsible for the outcome of repetitive systems, because they have significant influence on system conditions such as who to hire, how the work is done, and what investments to make.

Improve the Organization with Long-Term Thinking

It is all too easy to fall into the trap of becoming reactive and activity-driven in the world of product development. But succeeding with product development is not a random happening. Using long-term thinking, you can build in *system enablers* in your culture, organization, and technology that can reduce friction for initiatives and can accelerate learning. This will enhance your ability to spot, seize, and explore market opportunities.

Let's introduce a simple model.

The key to reading this model is to realize that the synergy of culture, leadership, and technology produce short-term agility. They work together. So if you invest in only one component and ignore the others, your payoff will be limited.

Leadership is your ability to seize opportunities and to turn information into action. This requires building up people and managers who are problem solvers and who take initiative when opportunities present themselves. Their efforts need to be supported by a system that is conducive to experimentation. Also, finding alignment across organizational borders must happen swiftly, without distorting the original problem. This is a challenge for tall hierarchies, which tend to have very slow decision cycles. A better approach would be to adopt a long-term leading strategy by building a culture focused on training.

Flexibility in organization is your ability to adapt structures, rebalance teams, change processes, and build skills to seize market opportunities. This requires making the leap from a familiar comfort zone to something new and unproven.

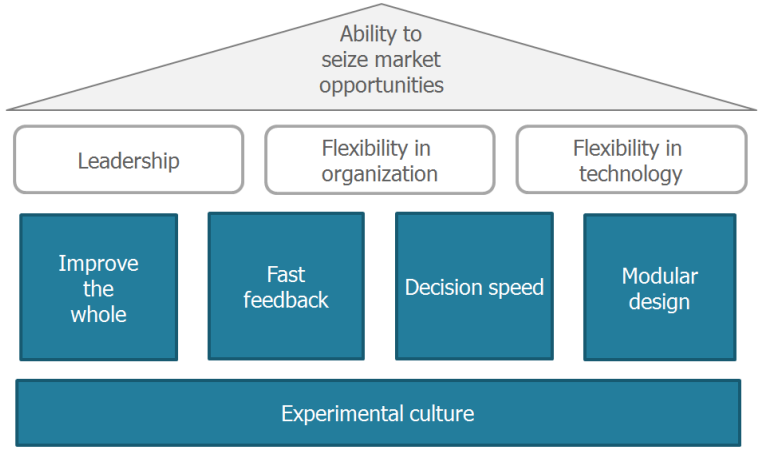
Making this leap requires *trust capital*, a certain amount of trust in your team's ability to deliver. Without trust capital, people in the organization will struggle during periods of uncertainty, look upon initiatives with suspicion instead of curiosity, and err on the side of safe bets. The good news is that management can choose to build up trust capital systematically. This is your buffer to handle risk and periods of uncertainty.

Clarifying your intentions, improving the whole, having regular conversations with people doing the work, and taking part in problem solving are examples of leadership behaviors that can help you build up trust capital.

Flexibility in technology is the ability to adapt your architecture and technology choices to late information.

If you find yourself in a vicious cycle of fighting fires and dealing with late surprises and late deliveries, what long-term leadership behaviors and system capabilities should you pay attention to? This is not always easy to see in the heat of the moment. To help you out, I've extracted five factors from the case studies in the book that you wouldn't see by simply looking at the Kanban boards. To notice them, you would have to observe events and behaviors for a longer period of time. What these five factors have in common is that they provide focus and direction for improvements and behaviors driven by the management team.

"All models are wrong, some are useful," statistician George E.P. Box wrote in *Empirical Model-Building and Response Surfaces*, and this is no exception. I've found models useful in helping to break out of vicious short-term cycles, so I share them with you in the following figure and table.



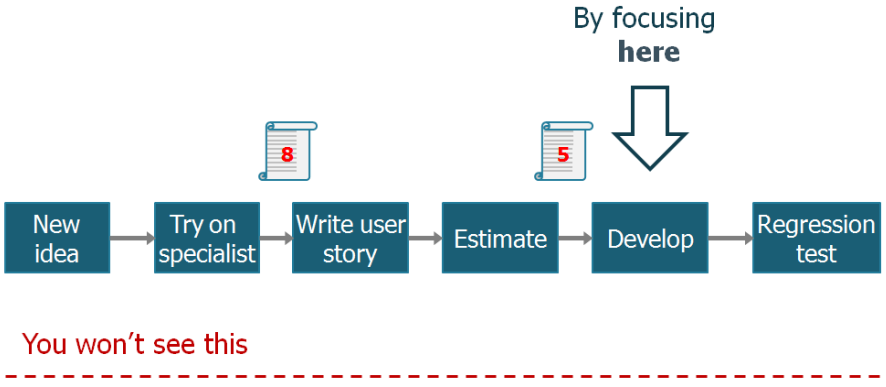
System enabler	Examples
Improve the whole	• Improve lead time through the full value chain, not just through individual functions. • Test working user scenarios instead of individual functions. • Reduce total costs, not just IT costs.
Fast feedback	Adapt your products to make them fit for your purpose by learning quickly through continuous feedback.
Decision speed	Keep up with market pace by detecting weak signals and doing something about them.
Modular design	Employ product design using loosely coupled modules.
Experimental culture	Shift culture from "No, it's not the way we do it around here" to "Cool, how can we find out whether it works?" using small experiments. It starts with your management team.

I have met people who push the above buttons either intuitively or out of experience. The trouble with intuition is that it's not transferable or teachable. I'm making the intuition explicit here to give some guidance on system capabilities a management team should pay attention to in the long run to stay competitive. I invite you to challenge your own ideas of long-term improvements. And if the only thing this chapter does is to push you to be more concrete with your long-term improvement ideas, to write them down and share them, that will already be a great leap forward.

The economics of the above capabilities and behaviors are such that they are cheap to implement during the early phases of product lifecycle and growth, but very expensive to catch up with later. That is why a management team benefits from paying systematic attention to them, especially during periods of rampant growth.

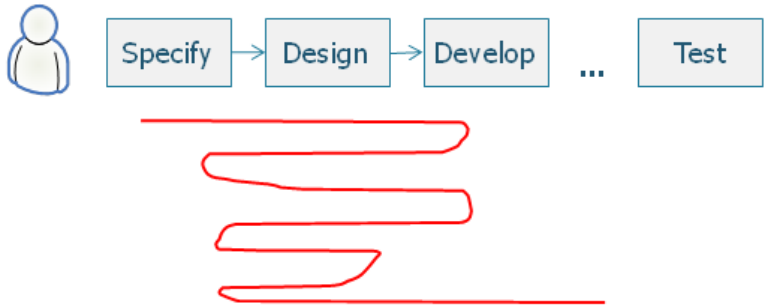
Focus on the Whole, Not Just the Parts

Improving the whole before the parts jars traditional thinking on how to manage, which is a legacy from the scale economy of the early twentieth century. As you can see in the following figure, the trouble with successful management of parts is that you lose focus on the whole, and this inadvertently becomes someone else's responsibility.



We touched on the importance of shifting behaviors in management from improving parts to improving the whole already.

A simple example of improvements that go unnoticed when perfecting parts is rework. Rework, often due to poor quality, hides between functions and is often invisible to the participants in a value stream.



In *Enterprise Kanban: Improve the Full Value Chain*, we chose the improvement perspective of end-to-end flow, and in *Using Kanban to Save a Derailing Project*, we used Kanban to visualize flow all the way to customer use, which enabled the parties involved to focus on solving one problem at a time.

Solicit Fast Feedback to Stay on the Right Track

No product is perfect the first time. No problem is perfectly solved by the first solution. Generally, some tweaking is needed once a solution is put to use for the first time. But the trick is not to wait for perfect information before starting. In any complex environment, waiting for the perfect solution means waiting until it's too late.

The trick is to leverage feedback throughout development to find out if you are on the right track.

For short-term cost-optimization purposes, it can be tempting to see test environments, test harnesses, simulations, and local prototyping abilities as cost-saving opportunities. But if the net effect is prolonged feedback cycles, they will produce devastating effects on your upcoming product development performance.

Feedback loops can help us understand if we are solving the right problem and if we are solving the problem the right way. Here are a few feedback loop examples. What is the time to feedback in your organization?

Feedback loop examples:

Product/Market fit	Who is my customer? What problems does he have? Are there different user roles involved?
Environment fit	What does the ecosystem look like where our product will be used?
Problem/Solution fit	Do my solution options fit the problem? What's the smallest thing that could possibly work? What product capabilities are important (performance, maintainability)?
Scenario fit	In what user scenario will our product work add value?
Product/Integration fit	How do we know if our product works as a whole? Are there any interference and compatibility issues?

Anything with "big" in it generally correlates negatively with learning from feedback. It's because there is a built-in incentive to stick with and to ratify earlier decisions rather than to act on new information, because change can be costly.

For both hardware and software, time to feedback is a leading indicator of innovation speed.

In *Enterprise Kanban: Improve the Full Value Chain*, early investments in automated tests enabled development teams and idea owners to shift focus and energy from validating what was expected to work to whether they were meeting customer expectations. In *Using Kanban to Save a Derailing Project*, time to feedback for both working solution and individual function was improved by faster test cycles.

Speed Up Time Spent on Decisions

An organization's ability to solve problems and make good decisions is key to keeping up pace. The fact that decision pace tends to slow down as an organization grows often goes unnoticed. In the worst case, denial prevails while voices of concern from real users or engineers are simply ignored instead of productively addressed, and the project eventually fails. Getting the right information at the right time is crucial to your ability to make shrewd decisions efficiently.

When your organization becomes Leaner or more Agile, it will put pressure on your decision cycles. One telltale sign that a decision cycle needs improvement is stress building up in management teams and "discussions at work." As flow improves, problem solving and tradeoff decisions need to happen daily instead of at the end of projects or on a monthly basis. Resource-allocation decisions need to move from yearly budgets to quarterly or monthly. One of the solutions that will shorten decision cycles is moving to a more decentralized organization.

But making decisions efficiently isn't the only thing that matters. Communicating the decision coherently so that people understand why it was made and what it means to them is a challenge, too. This step is often rushed through and taken far too lightly. If communicating what a decision really means is rushed and left up to different members in a management team who are bound to have their own takes, it can breed mistrust and a feeling that management is disconnected.

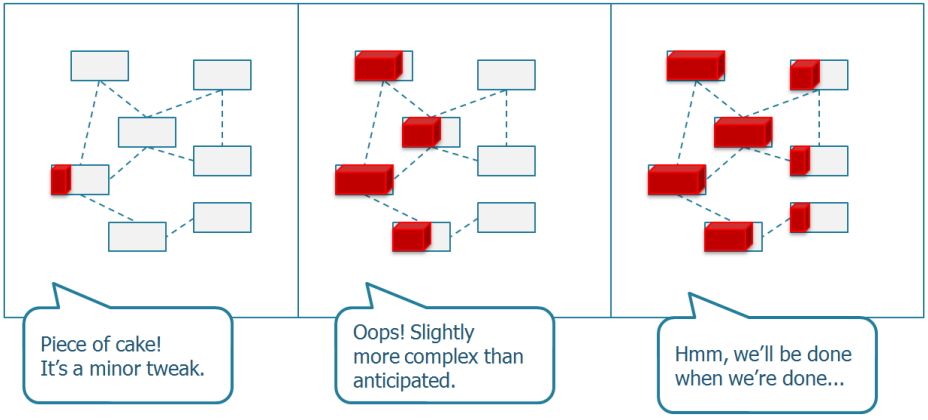
As a saving grace, however, there are a few things that can help out:

- Treating decisions in product development as tradeoffs rather than as absolute facts, and being wary of situations where you can't justify the tradeoff.
- Trusting first-hand information. The people closest to the problem often have the best view of the current situation.
- Using visual information. Having a shared view can greatly improve understanding of the current state.
- Fostering a "bad news first" environment. An environment where weak signals are taken seriously and problems get solved as a result, instead of one in which the messenger is ridiculed and shot.

Adopt Modular Design

We can do several things to build flexibility into our technology. One thing we can do to both give us a faster time to market and reduce the number of late surprises is to *modularize* our products with loosely coupled dependencies.

Modularizing products can help you simplify them by clarifying the exact purpose and function of each module. It allows you to control your dependencies instead of being controlled by them. It helps reduce the ripple effect. As you can see in the following figure, a small change in a product with unknown or hard dependencies can quickly ripple through the architecture.



Dependencies matter because they indicate risk, also known as changes that need coordination. By modularizing your product, you define a clear purpose for each module, and your dependencies become explicit.

You also have the choice of designing away the need for explicit coordination by coupling the dependency loosely. For example, if a change in Module A rarely requires a change in Module B, then the need for explicit coordination can be relaxed and downgraded to an as-needed basis. This will speed up flow.

To summarize, a modular design has the following merits:

- Knowing your dependencies up front, instead of being surprised by them
- Decentralized decision making within components
- Faster flow of customer value
- Clarity of purpose for each module
- Component reuse (preferably extracted from business needs)

This can in turn reduce the amount of coordination and planning overhead needed to manage your product development.

Cultivate an Experimental Culture

In product development, above all we need to continuously challenge the status quo. If we don't, we risk getting stuck in old processes, rigid structures, and obsolete tools simply out of familiarity and habit. We may never discover the fact that our outdated methods could be replaced by something simpler and smarter. As Pascal Van Cauwenberghe, XP expert and inventor of the Bottleneck game, puts it, "Software companies get into trouble by consistently choosing the simple decision before the hard."

The ability to learn is an essential competitive advantage. What does it matter if we know all about desktop products, when halfway through the project we realize that what we need is the mobile platform? If we have the right experimental culture, learning an insight late in the process should be seen as an opportunity, not a problem.

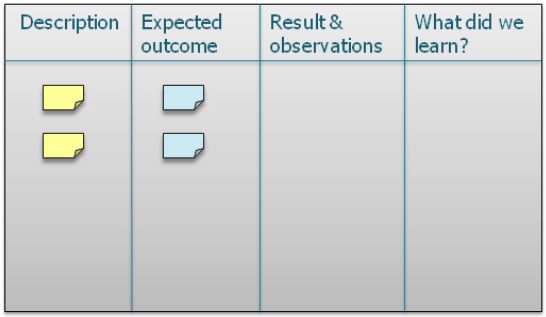
What behaviors drive a learning culture that promotes initiative? The easiest thing is to keep doing what we have always done, right? Have you measured initiatives in your organization lately?

Training management teams to improve on using experiments is one way to start unlocking system-level improvements across the full value chain. Since management decisions tend to have a significant impact on the structures in which people work (organization, processes, reporting, resource allocation, salary, and rewards), invoking management teams to challenge and improve at the system level is a natural step. So how do we train managers to learn?

The answer is by doing it! A simple way to train yourself to apply a learning behavior is by keeping a shared experiment board with your management team. Of course, the board is less important than the behaviors you seek to master:

1. Try ideas out, and try them quick (instead of promoting ignorance).
2. Talk about where you are and where you need to go (reflection).
3. Learn empirically using facts and observations.
4. Strengthen both critical thinking and curiosity.
5. Take responsibility for improvement ideas.

Let's look at a simple example, a visual experiment board used by a management team.



An experiment board helps a team to:

- Keep a shared picture of the experiments running, kept, aborted, or sustained.
- Clarify intended outcome. When we know what we should expect up front, we are better able to understand the actions that produced unexpected results.
- Separate observations from learning.
- Discover when we have new insights—new information that was not expected. Based on the expected outcome, we can learn when something unexpected happens.

The number of experiments you can run is up to you, but make sure each experiment has an owner. No owner = no experiment.

Many of the changes coached in this book were run in a similar manner. For example, starting Kanban was itself an experiment. We tried it out for a period of time. We then asked both the team and the manager what was valuable, what to simplify, and what to retain.

You don't need to write a post-it note for everything you want to try. By training people around you to define the experiment first before jumping into action, you are setting them up to learn from the outcome, which is often more meaningful and valuable than checking off improvement actions.

Key Points to Remember

All the case studies in this book started with managers who refused to be victims of their circumstances. They wanted to make a difference. While Kanban helped them see and focus their improvement efforts, the initiatives of the people involved made these improvements happen. Freeing up space

for improvement efforts couldn't have happened without changes in management behavior. Key changes included:

- *Evaluating output*. We achieved three things by shifting management's attention away from executing detailed plans and toward evaluating and improving output. First, managers focused on quality and flow efficiency instead of resource-usage efficiency. Second, and even more important, managers began to question and reflect on the effectiveness of the current way of doing things. Third, we now had transparency of the current state, which is the stepping stone to impacting the future.
- *Optimizing end-to-end*. By shifting focus from improving the parts to improving the whole, we can begin to recognize factors outside IT to improve on—for example, the front-end part of the value stream. By reducing time to market and percentage of features used, we can reduce total costs instead of the costs of individual functions.
- *Talking to people*. Have frequent conversations about where you're going and why. This helps nurture thinking and responsible people. It also builds up trust capital for your intentions, which is vital to have during times of uncertainty.
- *Leading using long-term thinking*. System enablers beyond the span of control of individual teams produce short-term agility. By making small, sustained investments over time in culture, technology, and leadership, you can spend more of your time in flow mode and grasping opportunities.

Finally, it's not the processes, the planning, or the execution that drives improvements and innovations. What it really requires is the initiative taken by the people involved. An experimental culture among the management team is crucial to help make the leap from *preserving what is*, to *looking at what is possible*.

Okay, that's it for now with theory. Let's walk through our case studies and see what really happened.

CHAPTER 2

Enterprise Kanban: Improve the Full Value Chain

It's easy to lose focus on what is important in software development—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.

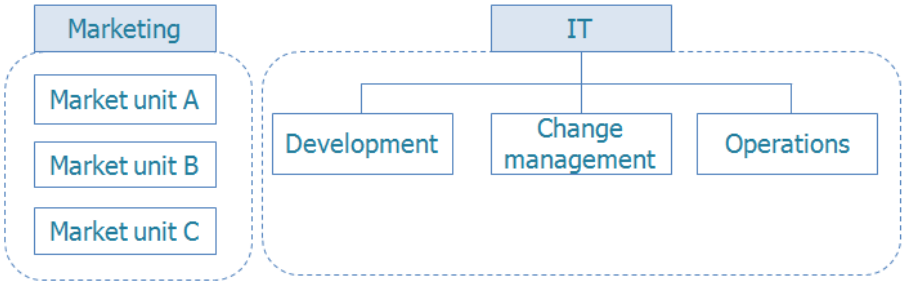
Very few companies start off improvements with a clean slate. They carry legacies—people, technology, roles, culture, market share, and processes. The legacy proves the company was successful at some point, but now results in heavy processes and structures, which slow down productivity and make things hard to change. There is a lot of bureaucracy. 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: Improve Time to Market at a Traditional Company

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.

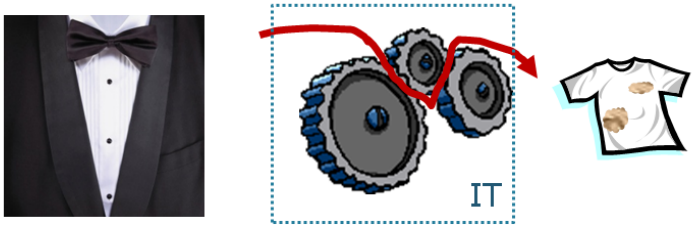
As you can see in the following diagram, the marketing department at Company H has three units, each targeting a specific market segment. IT is also split into three units: development, change management, and operations.



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 in order 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 the things that made the product unique to the developers, 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* if they felt ready 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 are 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.

It Is 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.

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.

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 standup meetings. (We cover more about this in *Focus Behavior with the Board*, on page 37.)

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.

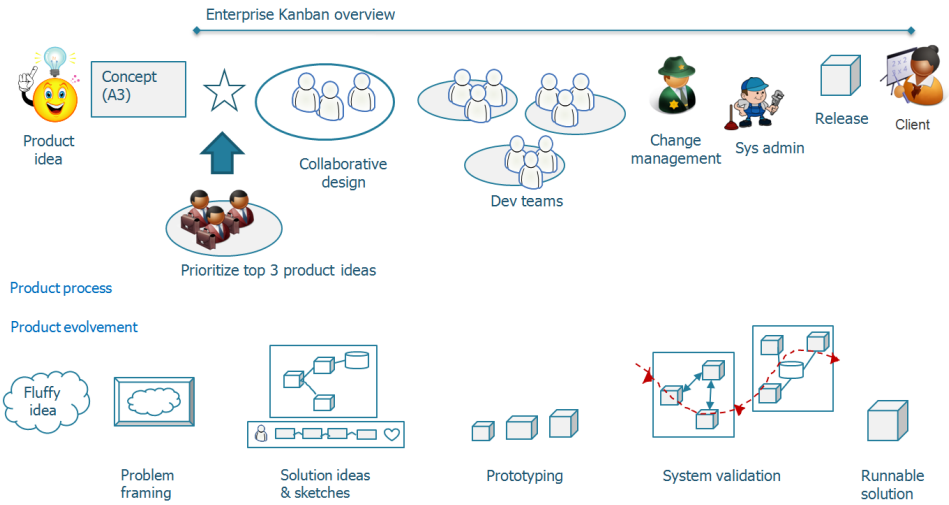
Reintroducing Sprint Planning

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 did not 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" x 16") piece of paper. With concepts, it is 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. We'll discuss this approach in more detail in Appendix 1, *Introducing Concepts*, on page 99.

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 is nothing to hand over to anyone. We think you are the right person to handle it because you are 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 which 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.

Don't Come with Pre-Decided 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 pre-decided by the time of the meeting. "Why am I here to give you ideas? It seems

Don't Come with Pre-Decided Solutions!

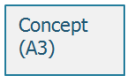
like you've already decided on the solution," a developer said during one such meeting.

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

Work with the Kanban Board

Now that we are 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 is placed on the Kanban board.

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.



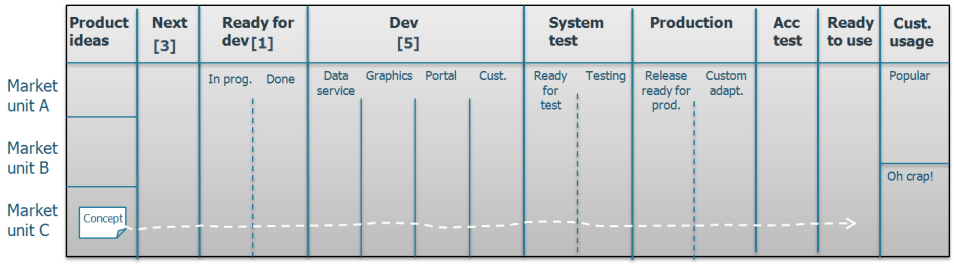
Concepts – marketable idea, tracked on Kanban board.

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.

Let's take a look at the board.



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 29, 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.

Acc. Test

Validation of the final experience by the concept owner.

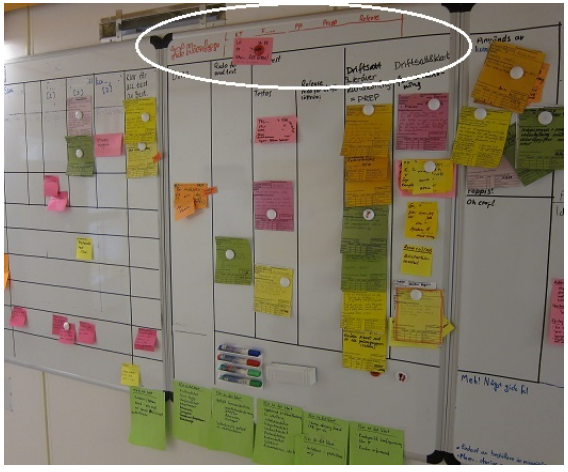
Ready to Use

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 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 is rarely used. It is still on the board mainly to assure the teams that if they raise a serious concern, they will be heard. This section is actually 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 had 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 vs. 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. Therefore, spending too much effort estimating the cost side of the equation is not effort well spent because you are 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 are 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.

Second, *Time through the system = effort consumed*: Use estimated lead time for the product idea to learn how much effort it has consumed.

With the cost range covered, focus can now be shifted to the value component of each product idea to make the decision on 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 35.)

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

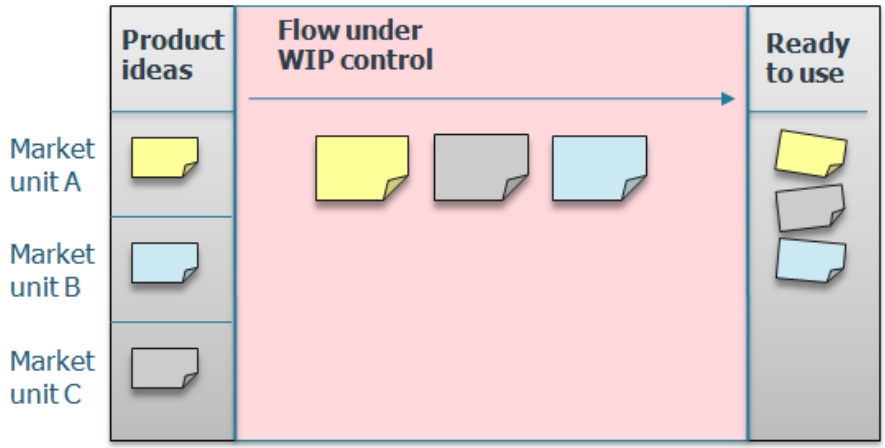
Match Estimate Effort with the Decision

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

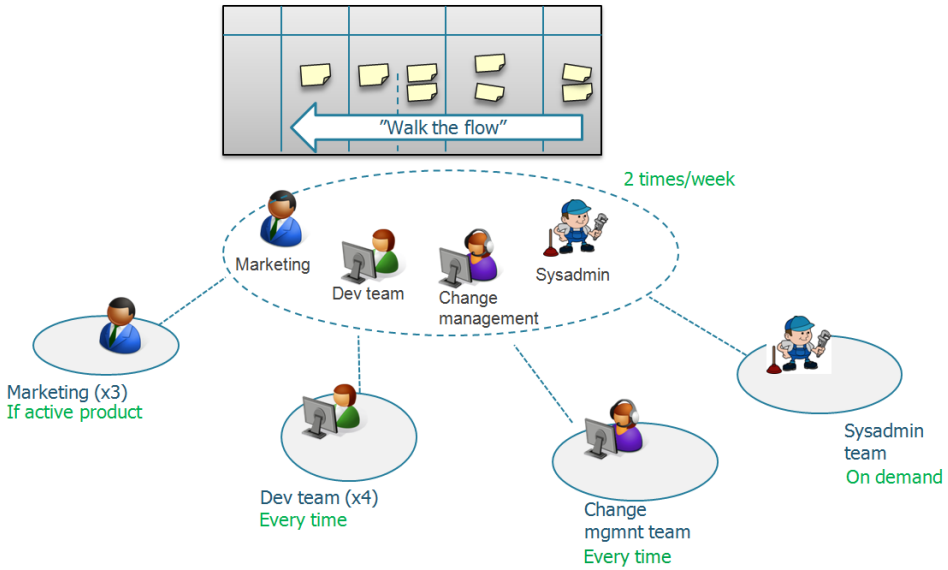
A question that surfaced early was what we should do about other work that was not directly related to the Kanban board. To avoid micromanagement and overloading the board, we decided on a decision rule to help organize the board:

- 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 standup 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.



At first, we asked for a (minimum) representation by each function at the standup. 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 standup." 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 make sure the standup meeetings ran smoothly, we stuck to a simple agenda. First, we walked the flow. The goal of this step was to find out whether there were any blockers 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 standup 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 are 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 are still holding standup meetings twice a week.

The idea of having a standup 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 standup truly becomes a decision-making forum and not just a vehicle for status reporting.

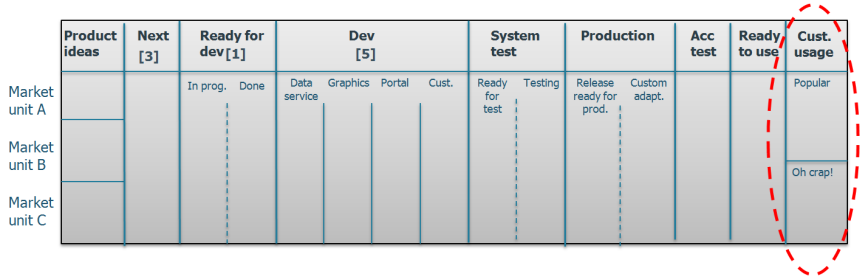
How We Continuously Improved

Take a look at the workflow again in *How the Process Worked*, on page 28. 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 are not finished when the product is released. We are done when the customer actually 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.



This became our most important feedback loop—were we delivering things of value? If the customer did not 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 did 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 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

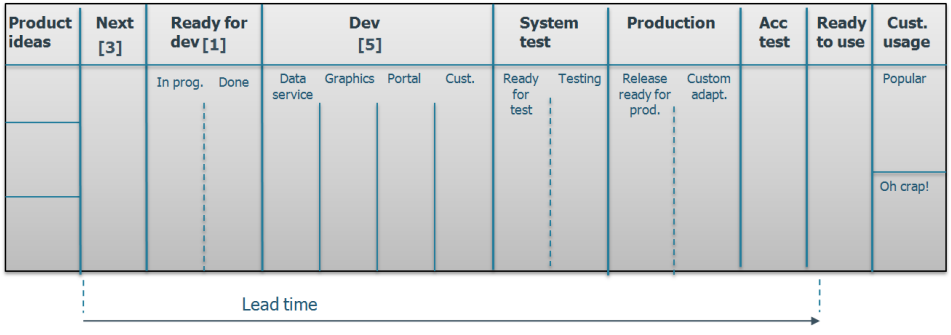
If for any reason you are not able to continuously validate the usefulness of a product under development, alarm bells should ring. Regardless of your choice of development method, this is your first clue that something will go wrong. Every new product needs at least one revision before getting it right—this 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). The reason for selecting these boundaries is because they are under our control. So extraneous actions such as how quickly the client actually 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* or 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 fixing 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 figure on page 42.

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 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.

The first time I reviewed these prospective new product ideas, I noticed that 40 percent of them did not 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.

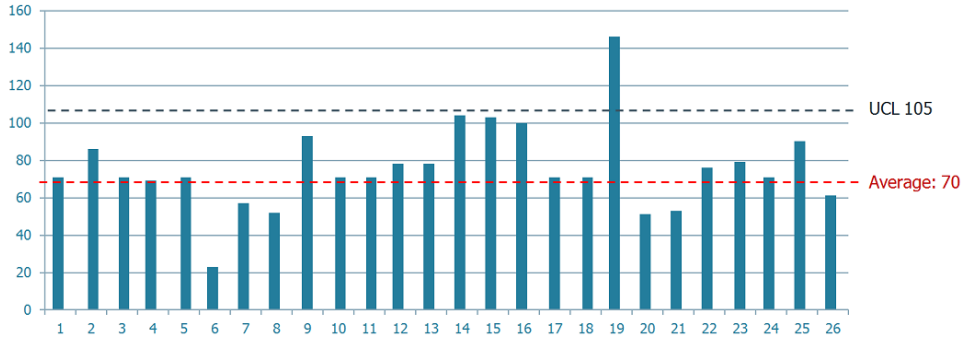
To fix this, the software team leads were given the authority to request that concept owners supply the missing information. Once we did this, we noticed a small but important behavior change—for the first time, the teams saw that they could ask for quality input, much in the same way concept owners 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).

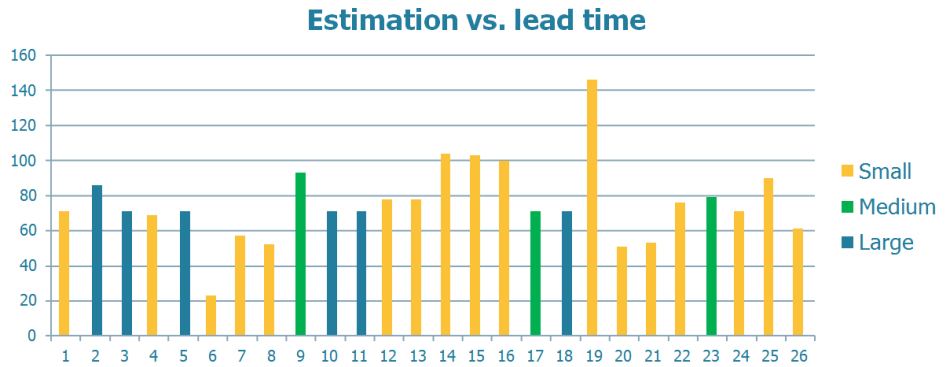
The following figure shows our first sampling of the delivery times for new product ideas. Each bar represents a delivered product idea, and the vertical scale shows the number of days it took for the product idea to become a shipped product (lead time). A UCL of 105 means 95 out of 100 product ideas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

I frequently hear the argument for needing upfront 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 will 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' upfront 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. 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 really only one estimation bucket

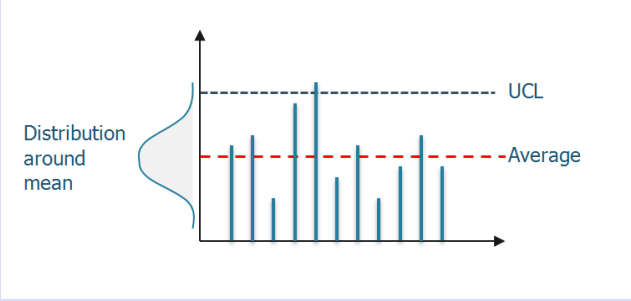
(or two, depending on whether you see the longest data point as a random event).

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 (2σ), 67 percent certain (σ), or 50 percent certain (average, not recommended). As a rule of thumb, I select UCL at 2σ, 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.

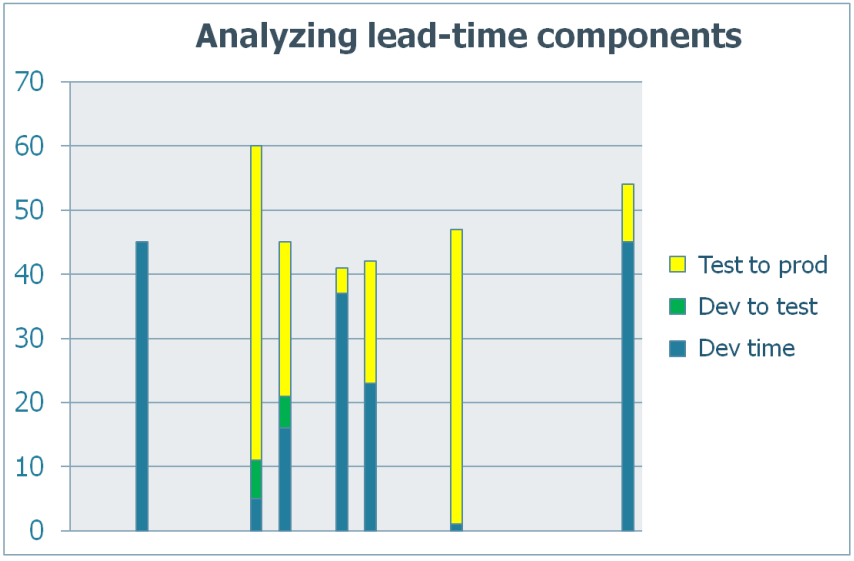
There are statistical ways to calculate the UCL, which I won't get into here. And there are hacks to estimate your UCL. Make a chart like the one above, with one bar for each lead-time observation. Draw a line across the top of the bars, skimming above the majority of them but cutting across one or two. 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 are 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 was not terribly 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

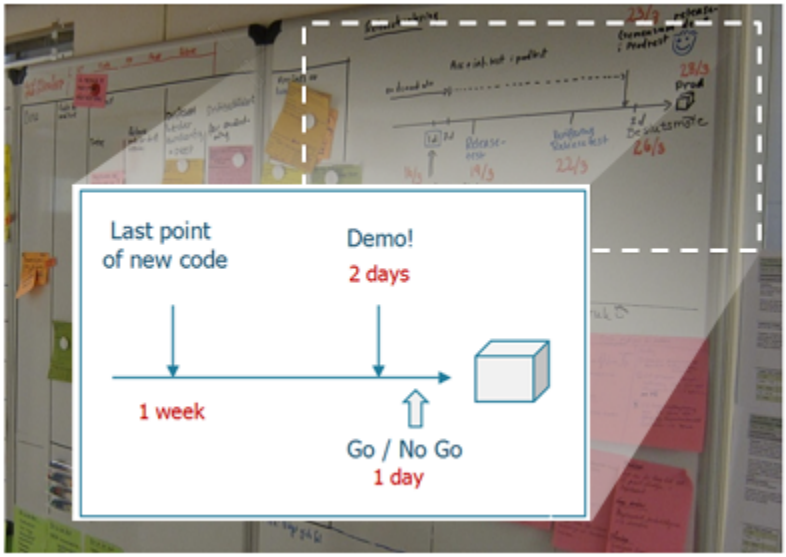
We had several examples of late changes pushed in very late during the release cycle, which made it hard to complete our system testing. We backtracked through the quality efforts required and concluded that this usually happens one week before the release. That was the time required to complete our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

When we looked at previous releases, system testing was often only partially complete. There were several reasons for this: market and time pressure, as well as 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. 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 does not 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 is not a good idea because it is 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 is perfectly documented, it is 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.

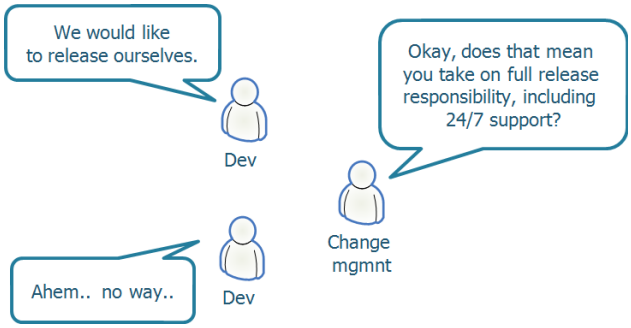
As we got more data, we saw that we could shave off a big chunk of the lead time if teams could release themselves; more precisely, if they could exempt themselves from the monthly release window. There are some advantages to this, such as the fact that we would virtually eliminate waiting time for system 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

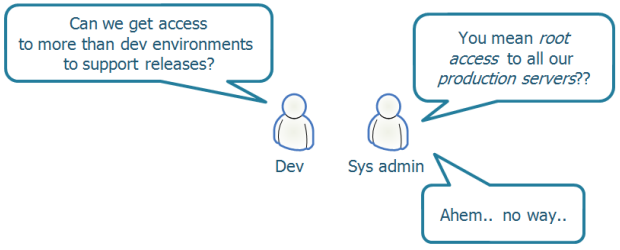
because of the monthly release window. If teams could release themselves and step out of the monthly release window, a one-day delay would be just a one-day delay.

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.

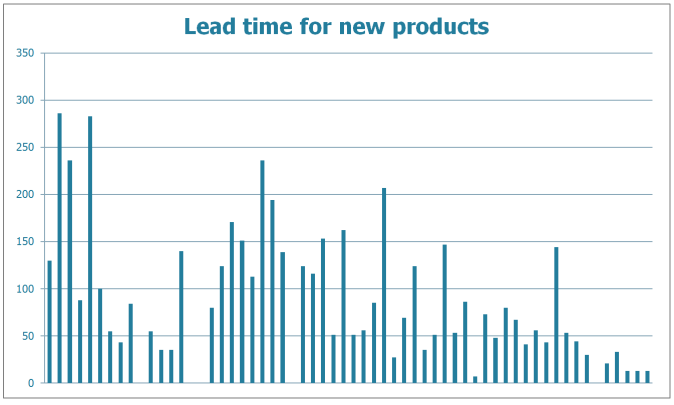
We also differentiated risk profiles. We decided it was okay to release at will for systems with few dependencies and good test coverage. We continued to release under the normal release window for systems with many dependencies and low test coverage.

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.

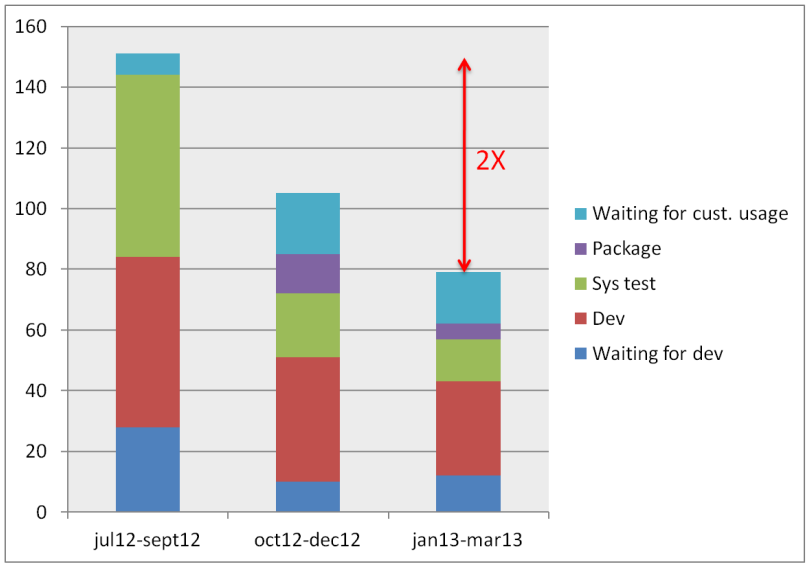
We moved through each step one at a time. It might not seem like a big deal that we finally got to a point where the development team was able to release outside the normal release window, but this would have been inconceivable two years prior.

How Lead Time Improved

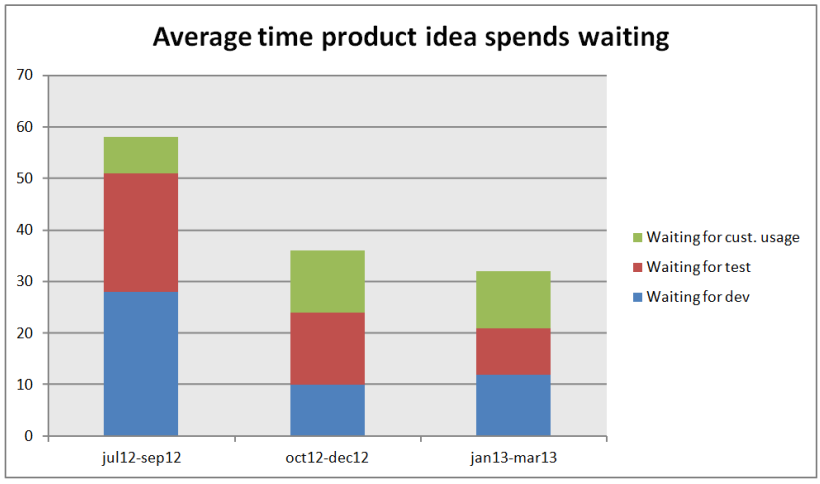
Did we improve? That's the most important question. All these changes don't matter if we don't have improvements we can point to. Let's look at some data, starting with lead time for released products.



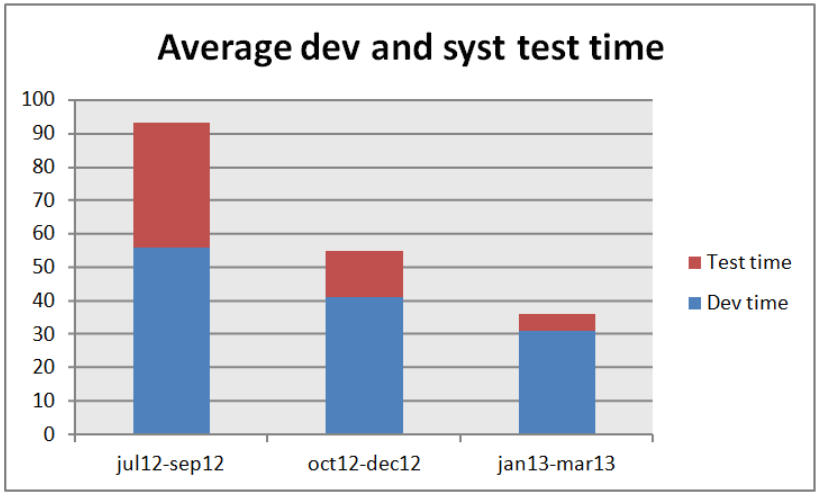
As you can see in the graph, lead time is trending downward over time, so efforts to reduce our time to market seemed to be paying off. You can get a better feel for this in the following graph, which looks at lead time for released products by quarter.



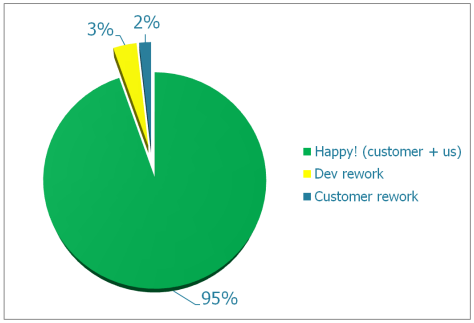
Products released through Q1 2013 got shipped roughly two times faster than in Q3 2012. This was great news, but we needed to know where our improvements came from. The following graph breaks down lead time into *waiting time* and *value-adding time*.



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 we were actually improving, we 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! There were two different ways something could fall into the Oh Crap! category: either the customer rejected our delivery (customer rework) or we didn't ship the product because we weren't happy with it (dev rework).

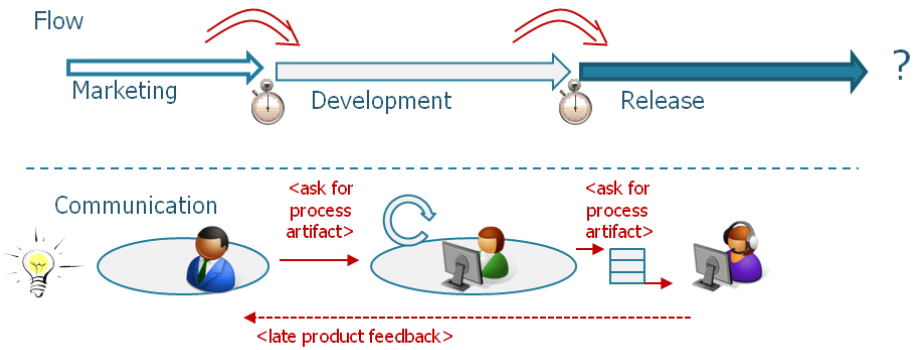
Have a Feedback Loop

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.

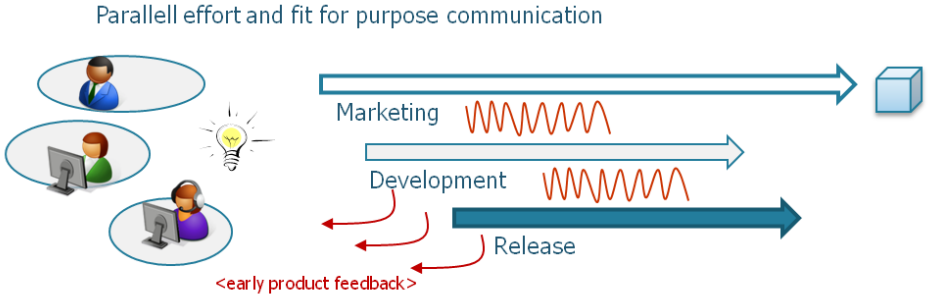
Comparing Now and Before

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 following figure 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 are 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. I have 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 is 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.

CHAPTER 3

Kanban in Change Management

Unfortunately, it's rather 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 *Enterprise Kanban: Improve the Full Value Chain*, 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+ 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 standup meeting. Last but not least, we also wanted to improve flow through our team.

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

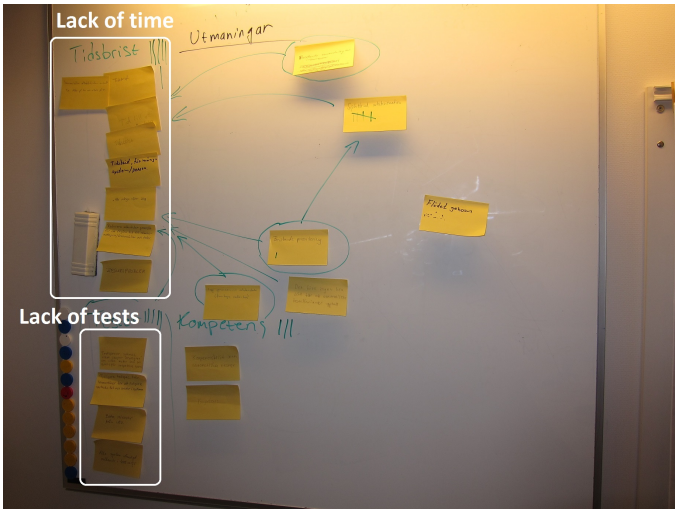
When we started, we had a very rough and chaotic board. We had tickets piled upon each other and quite a poor overview. At that point, we said, "We need to fix this," and we started to improve the structure and overview of the board. We also started asking questions about if work really was being done. It turned out that a lot of work that was piled up on the board was not actually being worked on. I remember on 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 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 about how to address the challenges.

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.



Two key challenges were identified at the workshop: lack of time and lack of testing during releases.

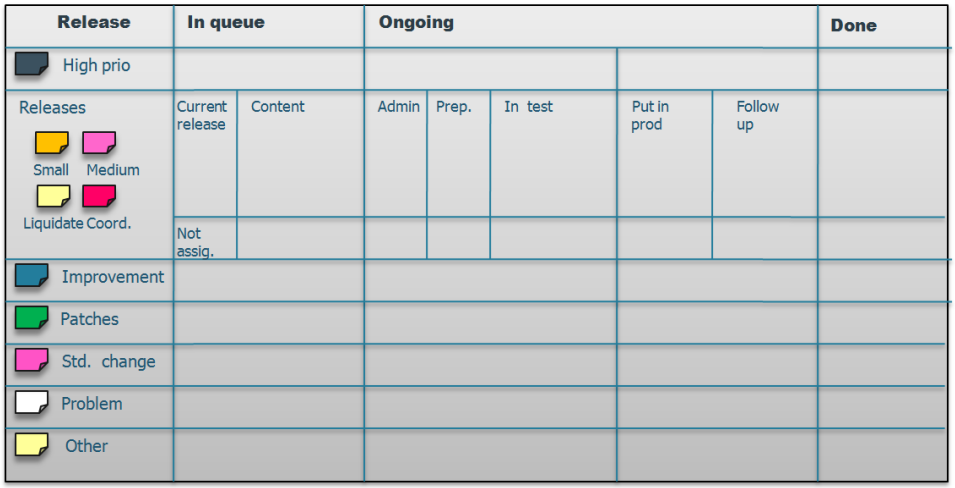
We discussed examples of how other teams use Kanban and the pros and cons of each approach. We asked the team to use simple thumb voting (see How to thumb vote on page 25) to tell us whether they would be willing to give Kanban a try. Once the team agreed, we could put up the board.

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 is where services are verified in production after changes are released.

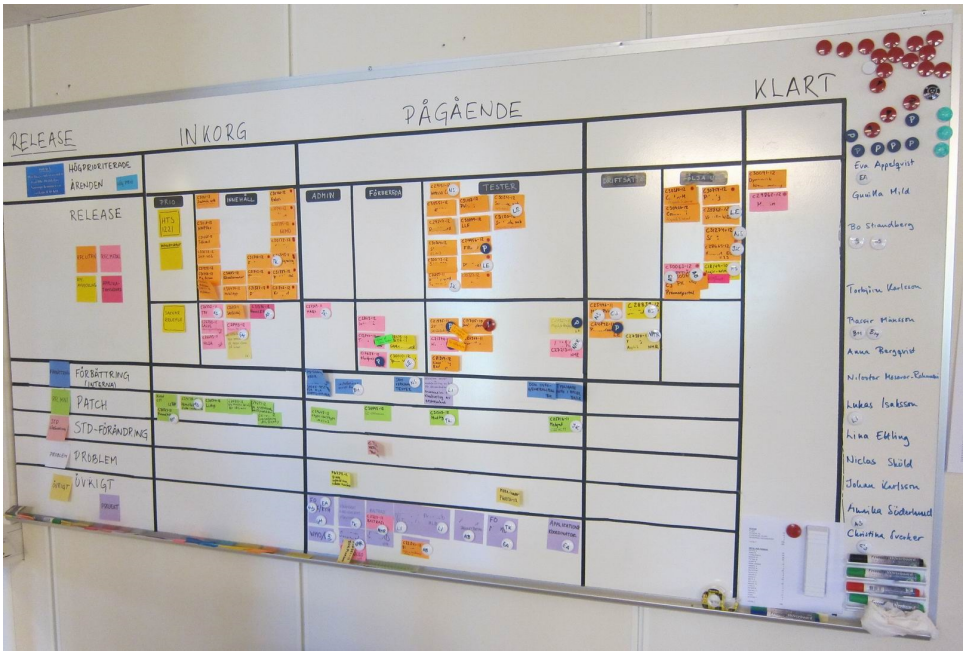


Let's take a closer look at how work flowed through the board. Each lane represented a specific priority and demand type. Tickets in each lane had a specific color to make it easy to visualize what was happening. We could see progress on non-urgent demands, such as improvements.

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: • Small - small changes • Medium - medium changes • Liquidate - remove from production • Coordinate - application change requiring explicit coordination
3.	Improvements	Improvements run and owned by the team
4.	Patches	Smaller changes to production, outside the release window

5.	Standard change	Standard change, impacting a single application
6.	Problem	Problems requiring investigations
7.	Other	Support to projects owned outside change management but which change management supports

Obviously, the board didn't start out looking this neat and pristine. The first version was so filled with tasks you couldn't actually *see* the board. The first version of the Kanban board also had a very different layout.



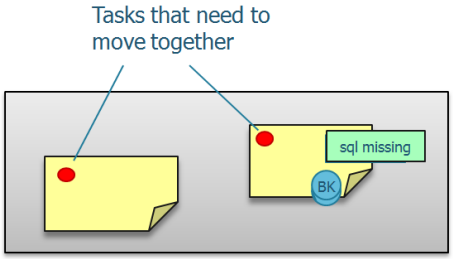
During the first week of using Kanban, we had trouble fitting all the work assigned to the team on the board. There simply wasn't enough space! To deal with this, the managers worked their way through the pile by asking "Is this active?" and then they would either remove or defer the item by putting it back in the prioritized queue or get it 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 simply 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 to see what to focus on and managers to 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 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 there is no behavior associated with the visual indicators, then they have no meaning.
- Usefulness: Are there 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 his or her 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 standup meetings a week where they shifted from what they needed to do as individuals to what they needed to complete as a team.

The teams ran standup meetings 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 there was an increase in the workload. Smaller requests coming directly to the team members were reviewed during the twice-weekly standups.

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 himself or herself. We tracked the number of *won't do* requests over time.

Expect Your Kanban Board to Evolve

It is 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" (backoffice) 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.

There was another challenge: the uneven distribution of skills. Some people did not have the experience necessary to be able to handle some of the more frequently occurring requests. The Backo person had to try to find a solution to the problem first 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 workload. However, we needed a way to document changes in the production environment and to keep track of all the details.

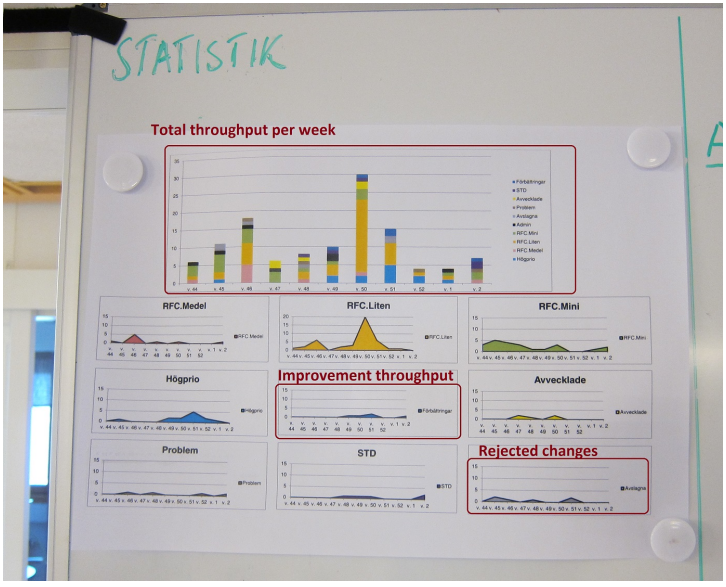
The ticket system turned out to do a pretty good job, and the team was able to use it as a documentation tool rather than as a communication tool. We learned to keep both the system and the board in sync by having the team update the issue tracker before moving tickets on the board.

How We Continuously Improved

After each release, the team help a small retrospective meeting to review what worked well and what didn't. We also updated the Improvements section on the Kanban board with ideas that came out of the retrospective. At the same time, the managers gathered data from the board and updated measurements for everyone—the team, the managers, and me (as the coach)—to review.

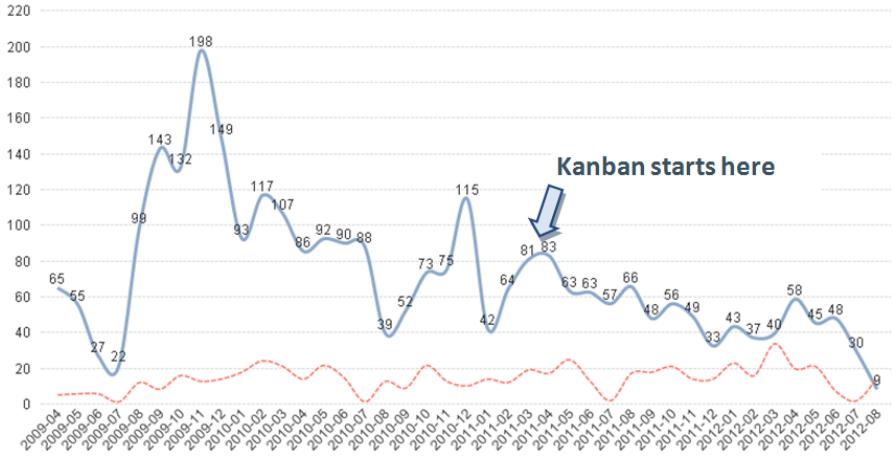
We collected three types of flow data: total throughput per release, throughput per work type (lane), and lead time. Collecting this data over time allowed us to spot trends as well as variations in these values. We posted the statistics on the left of the Kanban board, above the In Queue section, all to see.

The bars in the top chart in the next image represent total throughput. Note the large variation over time, correlating with the end of season. The highlighted chart on the bottom right refers to rejected changes. Not all change requests were well prepared, so some were rejected from the workload.



The highlighted chart in the middle reflects the team's improvement throughput (tickets designated as improvement initiatives). See how the team improvements 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 very 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 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 will 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 they were the ones consistently releasing 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 standup meetings. Before, the conversations were chatty; today, they are more focused, and the team now talks about deviations and what they need to act on. This is because the team feels they are 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."

There was an unexpected side effect of using Kanban. Adjacent teams, notably the system admins, began attending change management's standup 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 standups before, but we felt that they discussed items that were rarely relevant to us. Your stuff is more relevant. Can we join your standup?" 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 *fingerspitzgefühl* (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

There is a simple rule to follow if you don't know whether a tool is right for you: 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 is 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 actually 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.

CHAPTER 4

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 there is something 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. There was nothing left to trim, and it looked like the team was still going to miss the delivery date.

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

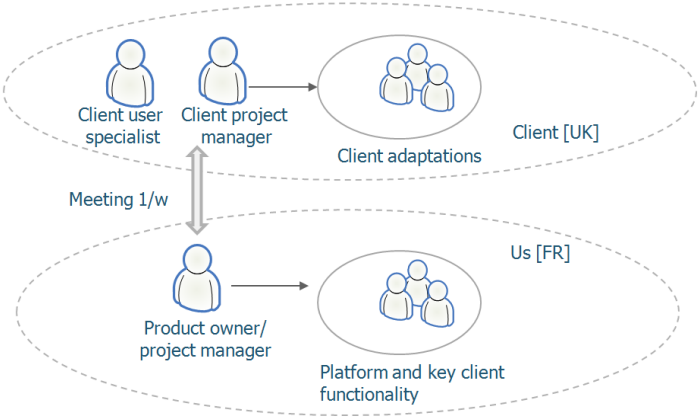
On a positive note—and there is always something positive—the development team consisted of very committed members and a positive project manager/product owner. The team had strong backing from management and weekly conversations with the client. The dialogue wasn't always pleasant, but at least both 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. While everyone had his or her own view of the situation, there was no common view or shared perspective.

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 was not up to the task of completing the project.

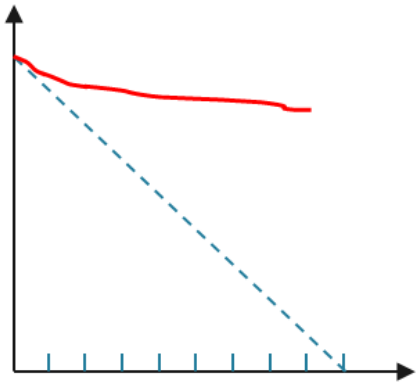
As you can see, the answers varied greatly depending on who you asked. Interviewing everyone involved didn't help us figure out what to address first, so we looked at how the team was working together.

Two development teams were working on this project. 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

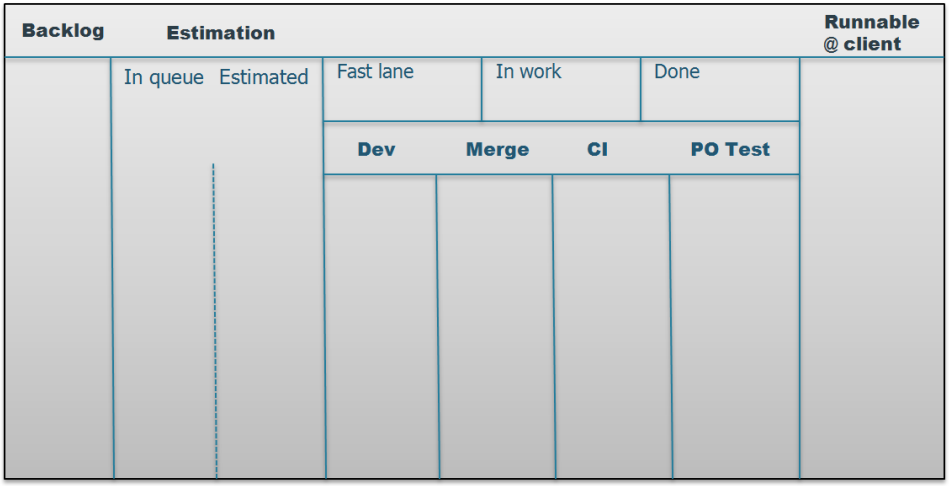
was working. There were several signs of trouble, such as the fact that the team was working massive amounts of overtime, often until one in the morning. You can see the worrisome pattern in the sprint burndown:



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.

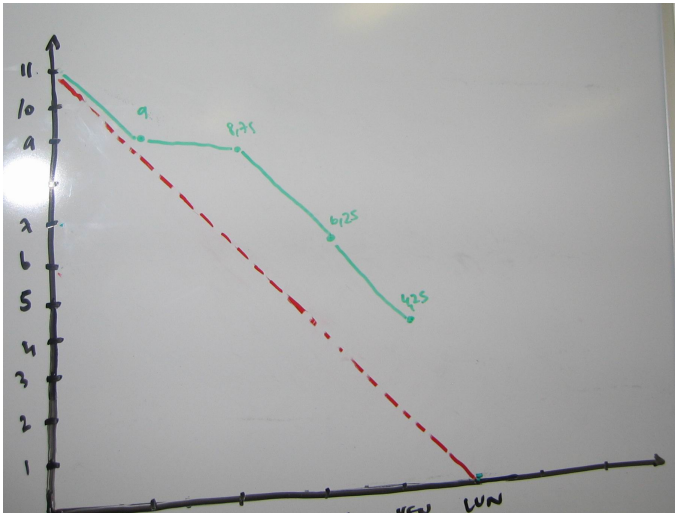
How We Got Started

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. 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.



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 the fact that there was now a limit on work in progress. We talked about WIP limits, which each team defines for itself, in *You Hold the Key to Your Future*. 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 could not 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

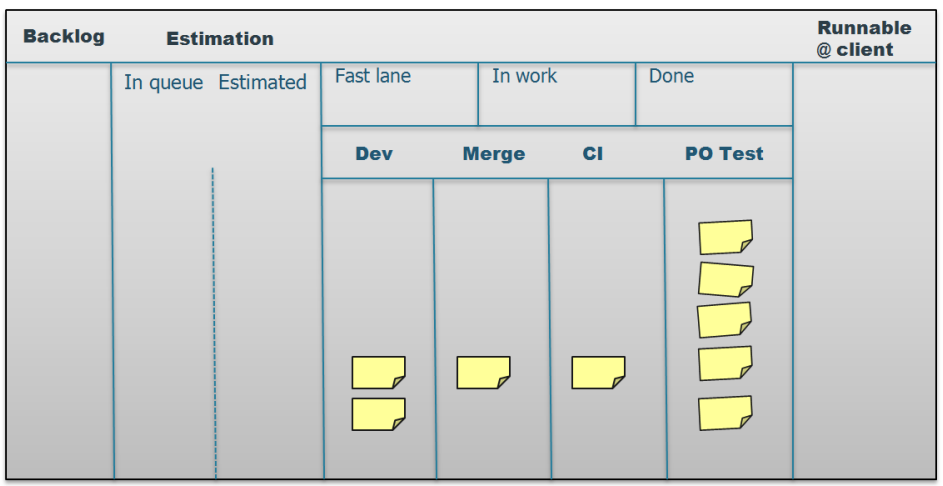
There are many ways to develop a shared view of the team's progress. It doesn't matter which indicator is being used, as long as it is meaningful to the people involved. Here's a tip: If the indicator being used to report progress is not being used to make decisions, pick something else that's more useful.

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 actually reflect the team's 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. The team was quick with development but struggled with testing and quality assurance. It turned out the project manager handled acceptance testing, but she could spend only half of her 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. However, 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 will 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 just had to wait one more week to get the desired feature.

With this decision, the team could focus. There was less emergency patching, less time spent on re-estimating tasks, and less time spent 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 quite turn out as expected, the client focused on validating the quality of the releases. There was a faster learning cycle 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.

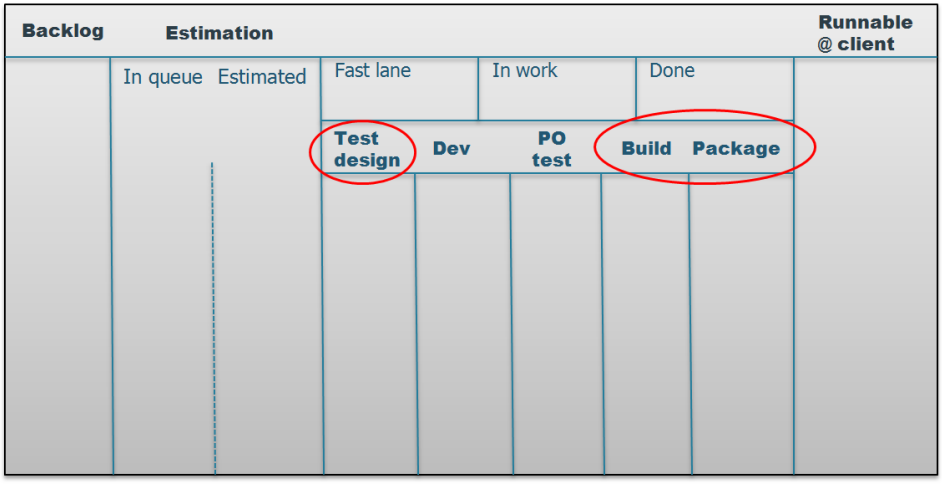
How Our Process Worked

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. 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 UK.



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.

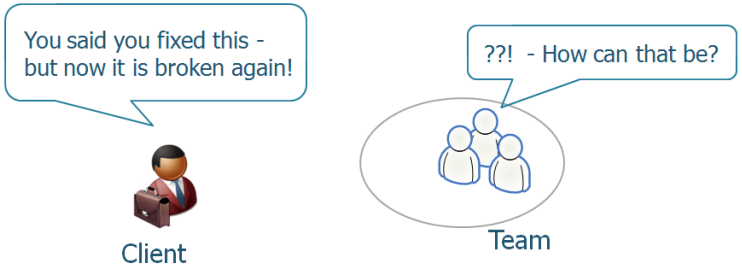
We compromised by focusing on baby steps. We designed for how we would test, and we used TDD wherever we could. We conducted manual testing where we couldn't write automated test cases. And finally, we committed to take one step forward each week toward implementing a working GUI test framework.

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."

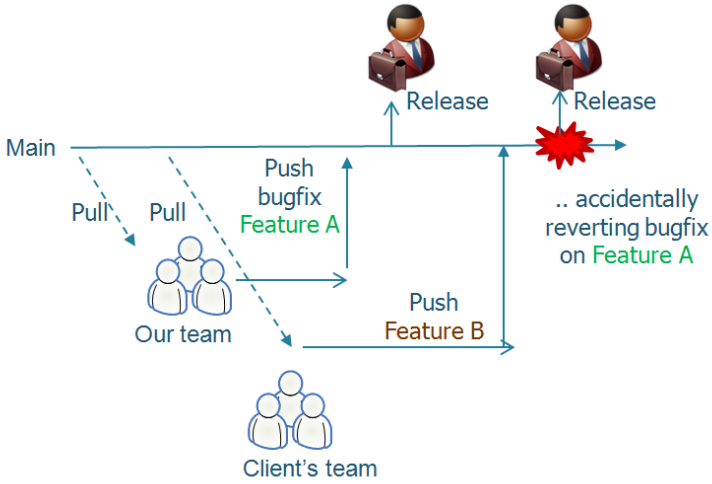
These small indicators showed that we were on the right track.

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 UK developer had inadvertently reintroduced the bug by committing old code.



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.

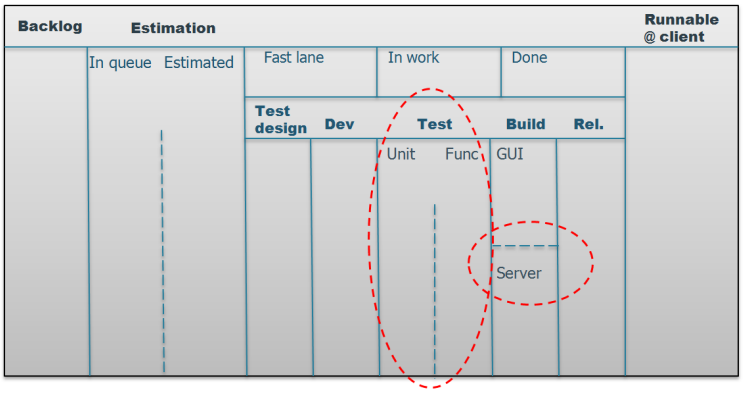
This is a case in point where you learn about how work is actually done under high pressure, as opposed to how it *should be* done as documented by some process or procedures manual.

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, 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 a very 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. 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 5-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 acknowledgement 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

There was no one single thing that 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.

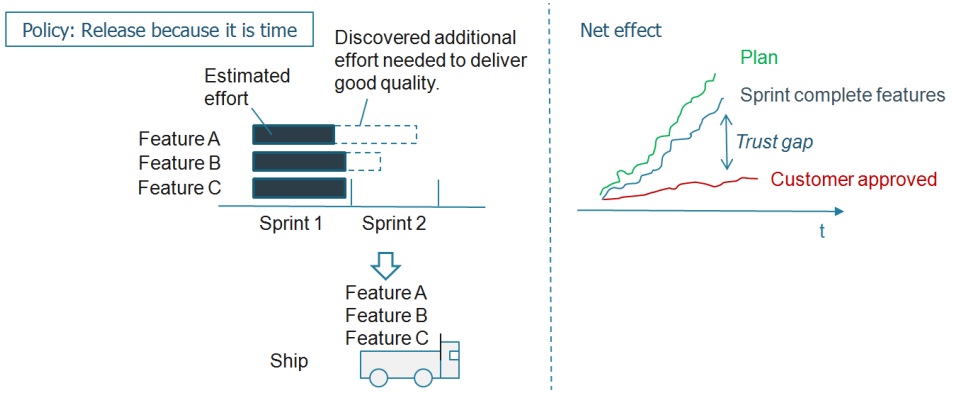
What did we do? We visualized where we had real flow problems and solved one problem at a time by focusing on quality. We wanted to get on that upward trajectory.

Why Our Scrum Implementation Didn't Work

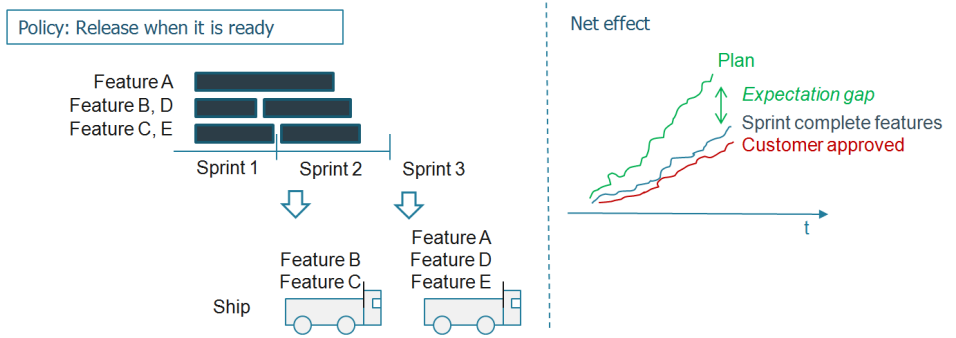
There are a few reasons why our Scrum implementation didn't work out, including rework and wasted effort, not refactoring the code, and the lack of cooperation between teams. Let's look at what we were doing wrong.

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 timeframe. 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 actually delivered. There was an invisible but growing trust gap between all parties involved.



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



Another problem was the fact 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. There was the sense 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, there is always more than one answer to a given problem. 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 are going to work in longer sprints and keep away while we do" would not have inspired a positive reaction.

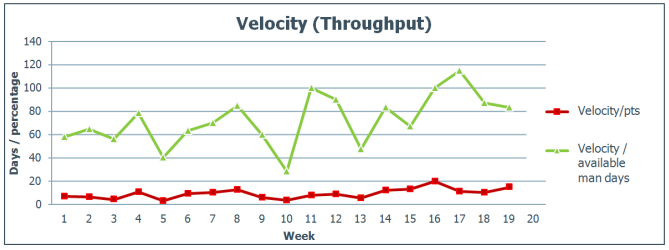
We could've changed our definition of *done*, except we weren't the only team involved. We had no control over the client's deployment process. If we changed the definition of what it meant to be done, the client was the one who had to deal with the consequences. We accomplished something similar by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

CHAPTER 5

Using Kanban in the Back Office: Outside IT

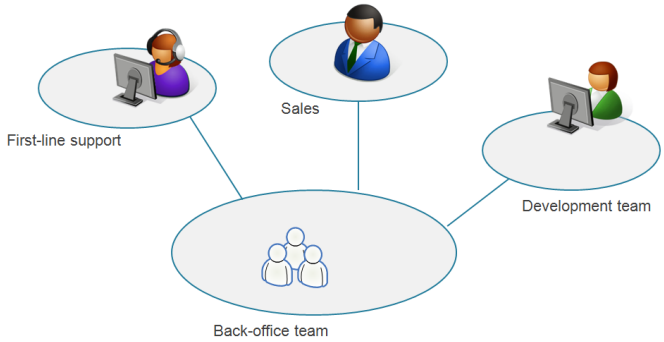
When we talk about change, it's easy to get bogged down with a specific method or technique.

We have 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.



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 quite 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—out-

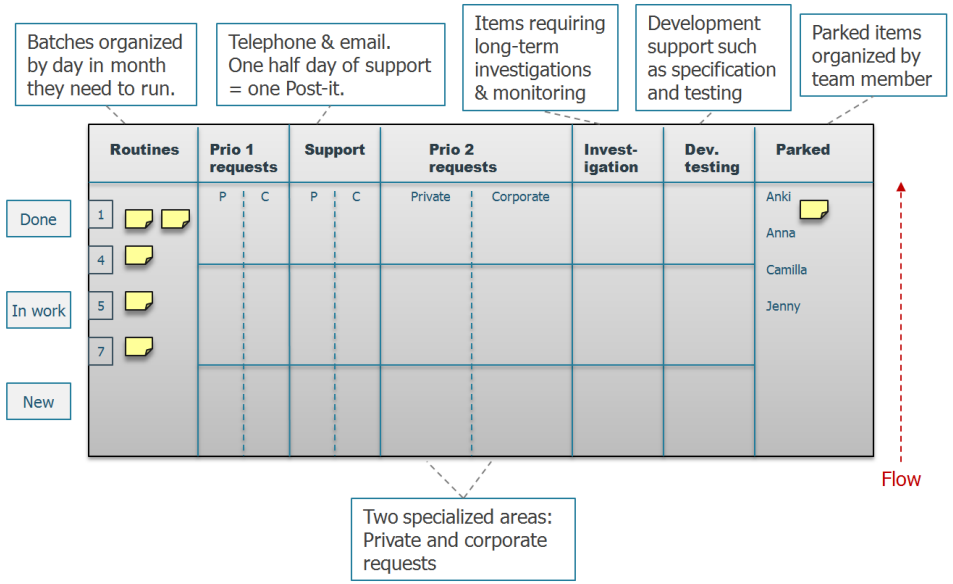
weighed those concerns. The team decided to give Kanban a try, and we soon had our first board up and running.

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 will 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.

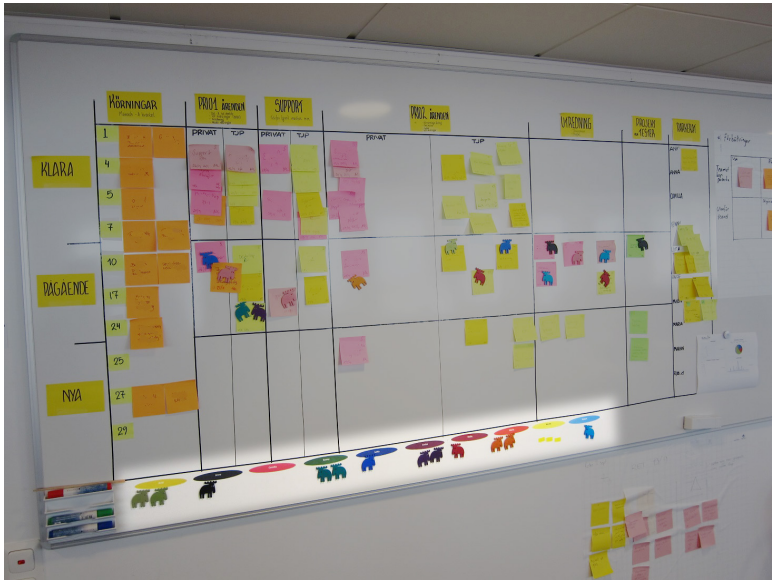
Demand Type	Column / Description
Routines	Some of the work was recurring. For example, "On Mondays, we have to run the payment batch." All these routines were kept on the left side of the board in a column organized as a (monthly) calendar. When the day in the month arrived, the employee would fetch 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

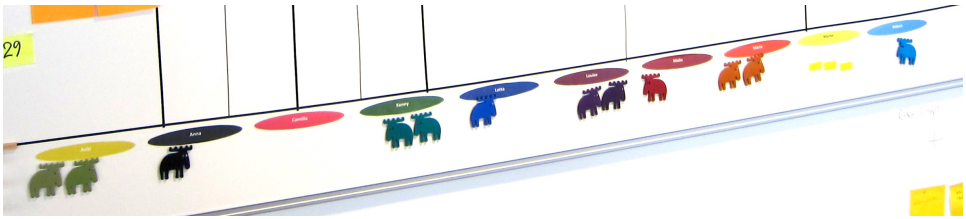
There is a challenge in maintaining a *parking area* for items that require external interaction: it's easy to forget about them. You can use one of three approaches to avoid this:

- Before you park an item, you should make sure that you have completed everything on your part first, because revisiting work can be costly.
- You should walk through all parked items in one of the weekly standup 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). Every team member received three elks to place on the board. If the person had all the elks on the board, then he or she could start new tasks until all the existing work had been completed.

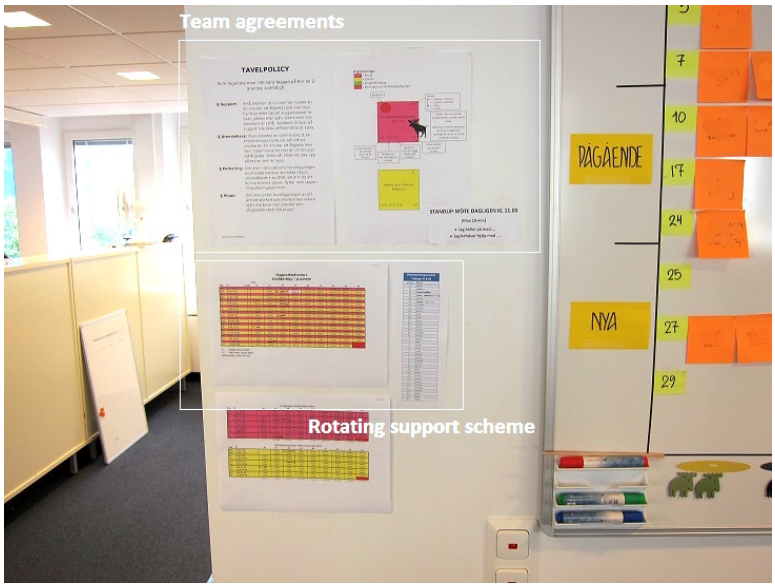


Using a limited number of physical tokens helped everyone to see who was working on what and to track work at risk of stalling. Work can stall for various reasons, including people being away from the office or being sick.

Before this team started using Kanban, support requests and small questions would just pour in to a random person 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.

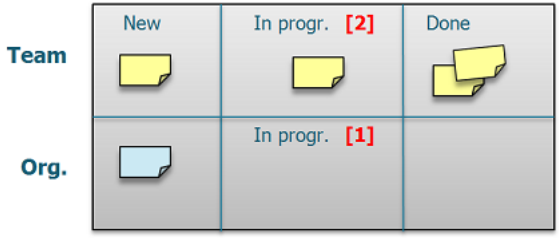
The members were positive about being part of the team. They liked the variety of work and often named good colleagues as one of the upsides of working with the team. With everyone managing his or her own workload, sharing responsibilities such as keeping the board updated and alerting each other when someone else was stalled did not 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.

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 standup 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 visualize ongoing improvements under an *improvement Kanban*, which we kept on the right of the main board.



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 really powerful. We will 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 really nagging issues in a short time, 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 is 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:

- 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. There was a positive vibe around this team that 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.

There are plenty of minor tips and tricks we can learn 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 are under stress, the activity around the board increases and the team really 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 really 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 just 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 couple of 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 timeframe 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.

Part II

Appendix

APPENDIX 1

Introducing Concepts

The Elevator Pitch

For one of my clients, 50 percent of the products developed never made it to customer usage. That's 50 percent of hard-earned contracts that never generated any revenue. Improving that rate was obviously critical to survival in the marketplace. Developing things that people want is more important than running faster.

To address this fact, we shifted our approach to product development. Instead of having successive roles (Customer, Business, UX, PO, Team, Operations) drive development, we gave this role to one single person—we let the passionate person behind the idea run with it all the way to the working customer stage.

That's quite a leap. The challenge was to keep the integrity of the initial idea intact all the way to the end user. We needed a way to train people new to product development to ask the right questions before engaging in conversation with designers, engineers, and architects. Enter the idea of *concepts*.

We did have some supporting infrastructure in place for the transition. We had ways to limit WIP and to prioritize ideas, since we couldn't afford to run with all ideas. The most interesting consequence of this shift was that our engineers went from feeling "like cogs in a machine" (routinely completing stories in sprints) to actively taking part in creating working solutions, accepting nothing less than awesome.

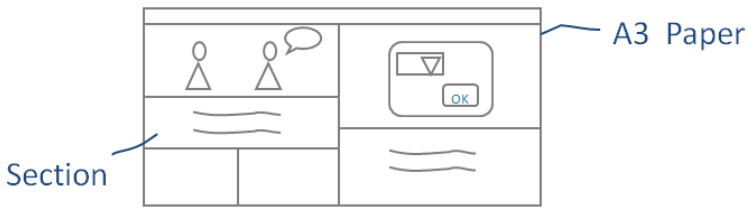
We called this approach concepts.

Concepts can help you:

- Preserve the integrity of the original idea.
- Increase the value-added time of your development team by training businesspeople to arrive prepared with the right questions.
- Avoid burning out product owners by sharing the workload and elevating the quality responsibilities of development teams.
- Train businesspeople unfamiliar with IT on how to enter into fruitful conversations rather than getting lost in details.

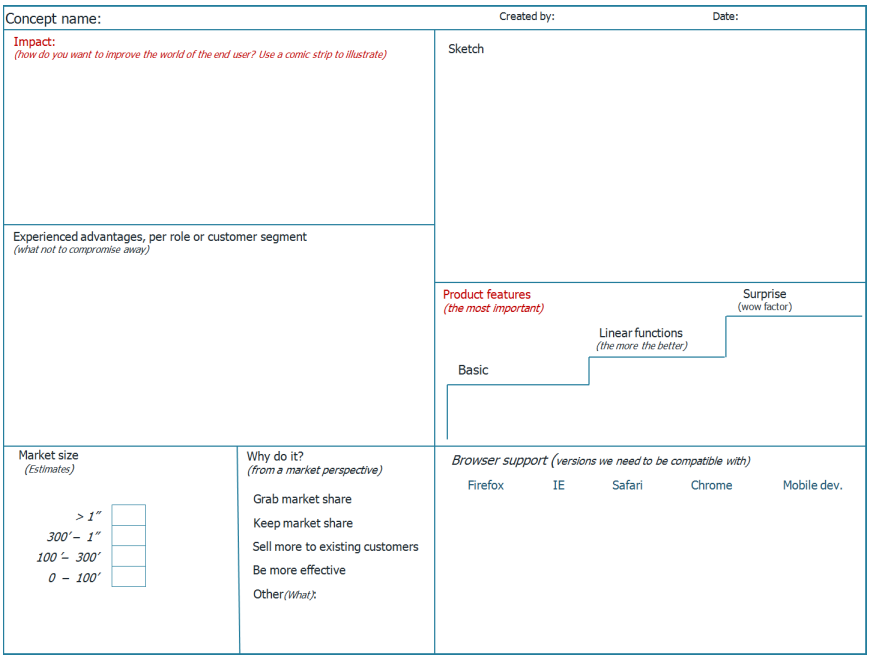
What Is a Concept?

A concept is basically an A3 sheet of paper (12" x 16") with predefined sections for questions that should be answered before you enter into a conversation with the development team.

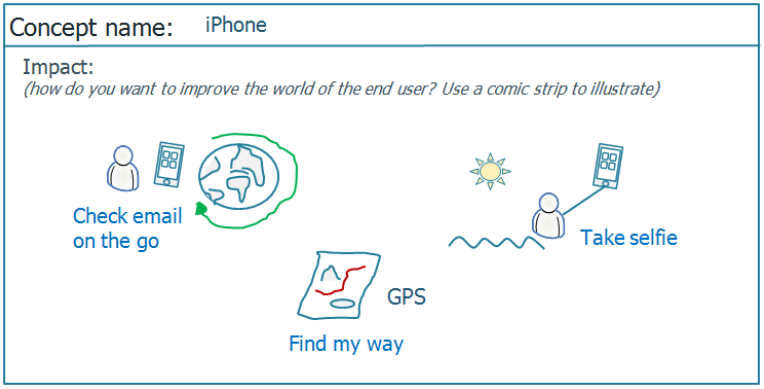


Think of a concept as a flexible minimum specification, on one page as shown in the preceding figure. The three questions to consider are as follows:

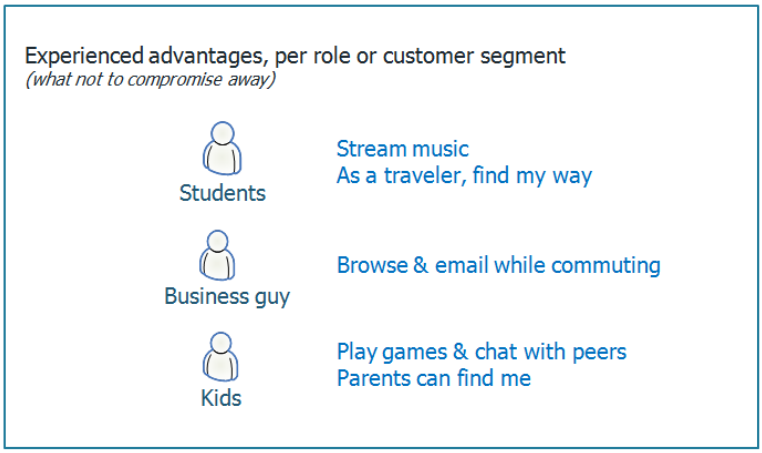
- What is the product's impact? (What gets better in the customer's world when he or she uses the product?)
- What are the product tradeoffs you're willing to make and what must you stand your ground on?
- How can you, as a stakeholder, arrive prepared for a conversation with the development team?



Imagine we were about to kick off development for the iPhone. Let's take a look at what a concept might look like.

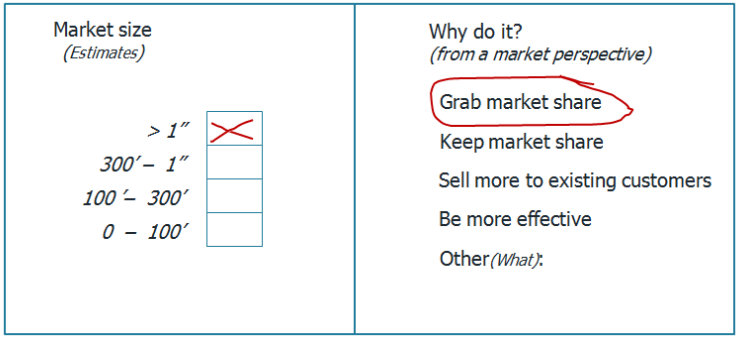


The first section is Impact. There we describe the impact we want to generate for the end users—how we would like to change their world with our product. The sketches are there to describe the problem (not our intended solution), including the real-life situation where our users would use the product.

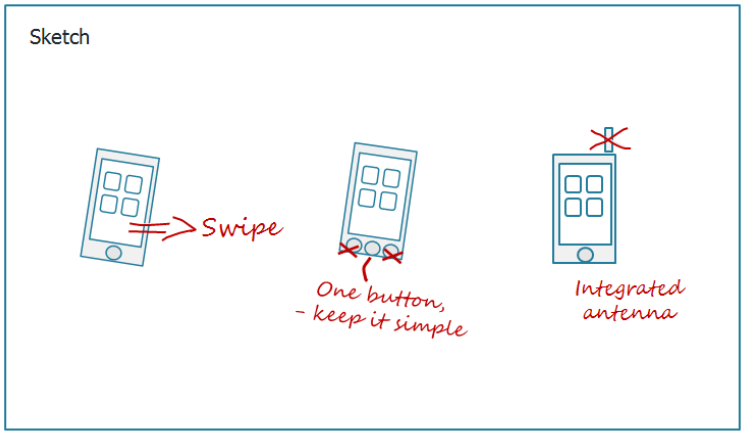


If there is more than one user group, we illustrate the value we intend to give to each user group in the second section. This can help us see at what point our product provides enough valid content to necessitate an update, if multiple user groups engage with it.

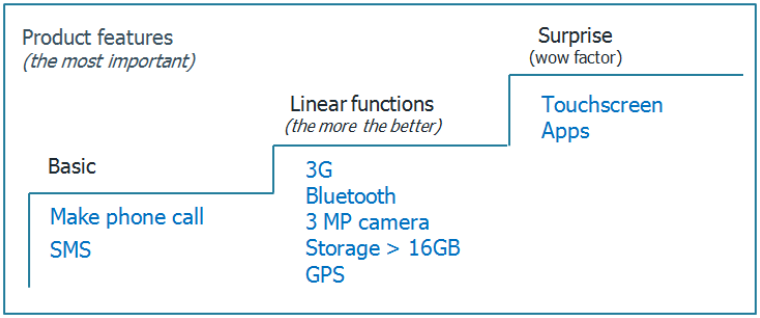
The third and fourth sections are simple indicators for the business case, helping us decide which concept to go with during the inception phase of product development.



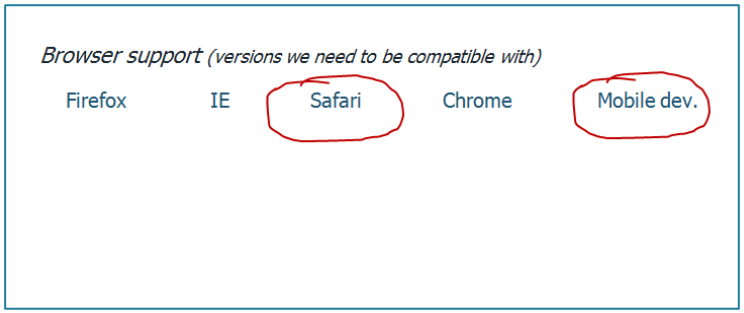
In the top-right Sketch section, we illustrate early design ideas. We use this section to demonstrate that we have some concrete idea in mind.



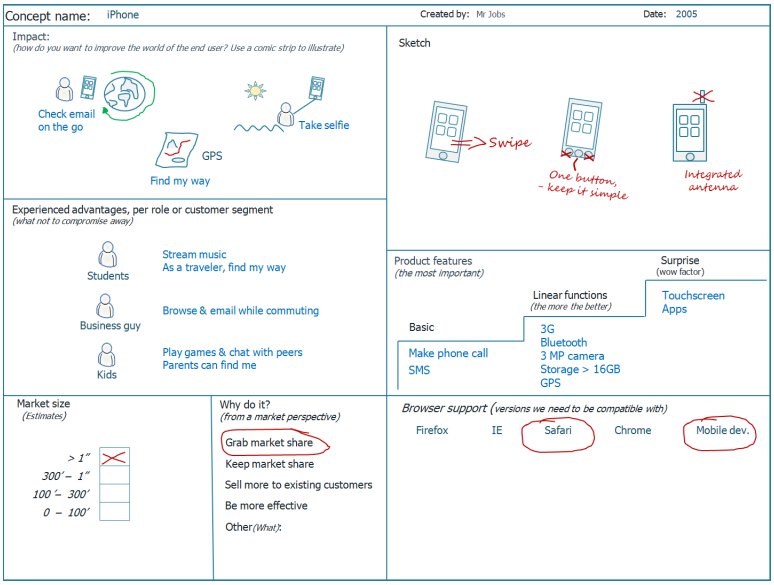
The Features section illustrates our view of customers' perceived value: Basic (product needs to have these features to be viable), Linear (the more features, the better our customers will perceive our product), and Surprise ("wow" features our customers didn't know they needed until they saw them). It's a simple overview based on the Kano model.



Finding the wow factors is actually quite challenging. Making sure that we have a good idea of what the wow factors might be is time well invested before jumping into development with both feet.



The final section, Browser Support, is really there to communicate risky tech assumptions. It could be replaced by nonfunctional requirements (NFRs)—for example, performance, which has caused repeated quality concerns based on previous experience.



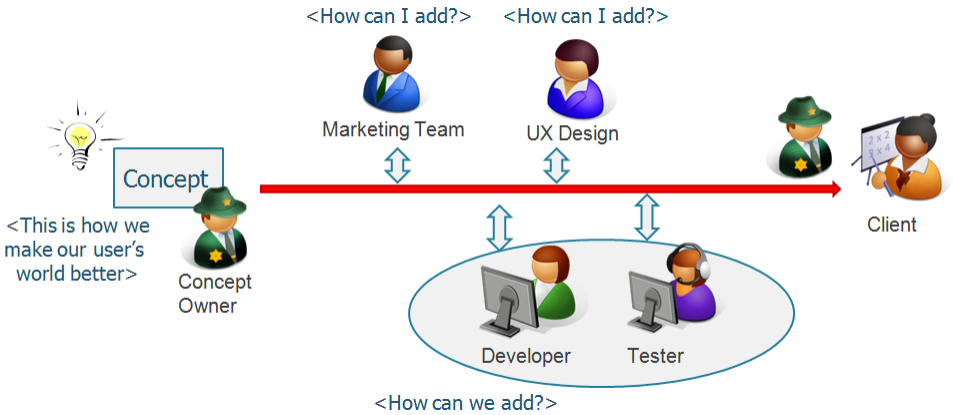
What's the Big Idea?

Concepts can help you start a focused conversation so you don't get lost in the details. If you cannot squeeze your product idea into an A3, you probably haven't quite grasped its essence yet, so more discovery work is needed. Concepts can help you avoid situations where fuzzy problems are handed over to UX, architects, and developers who have to reinterpret the problem to create the grand design—a sure recipe for scope creep.

Concepts enable people with good ideas, regardless of role, to develop new things. This person could be a salesperson, a developer, a consultant, a client—anyone who is passionate about an idea and who is willing to follow it through. No handover to a product owner is necessary. The passionate person behind the idea runs with it all the way to the satisfied-customer stage.

Avoid: many handovers.

A better scenario is one where one person guides the product idea all the way to customer delivery.



If You Want It, You Make It Happen

In the real-life cases where I've used concepts, my first comment to anyone who wants a product idea developed is, "That's a cool idea. Are you prepared to become the concept owner of this?" If the person answers no or with something like "uh, that's someone else's job," we stop then and there to avoid wasting time and energy on ideas that no one is passionate about. If you want it, you make it happen! The implied criterion is that you have to care enough to make it happen.

This is new for people who are used to handover ideas. With concepts, you follow your idea all the way. There is no handover.

Hey, Isn't That Too Expensive?

Are you worried that all this is going to be expensive? If so, take a look at how many of your products released actually got sold first. What would it be worth to you to raise that ratio? This exercise will clarify the question.

In cases where I've used concepts and we've had product owners, the function of the product owner was to choose which concepts should get developed and to make sure that they were added to the product.

How is a concept owner different from a project manager? The key difference is that it is not the concept owner's job to move the idea forward through the development process. The concept owner's key role is to manage the integrity of the idea before, during, and after IT to ensure that it has value to add in the customer's environment.

Remember that concepts are just a tool, not a silver bullet. Products are developed by people. A great tool can never supersede great people. The job of a great tool is to help skilled people find more value-added use of their time. Good people deserve good input. That's the idea behind concepts—to provide quality input.

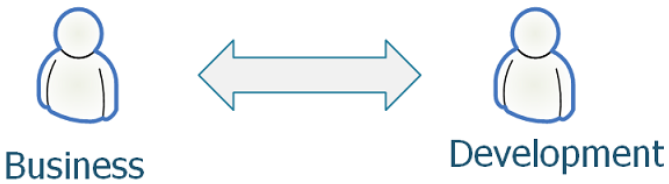
Concepts are not the only way to go. Here are a few other useful tools worth mentioning:

- Lean Canvas
- Impact mapping
- Story mapping

How to Get Going with Concepts

What if you wanted to try out concepts? It's easy. There are four basic steps.

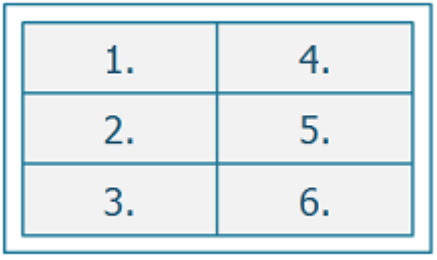
Step 1: Business and development meet and agree on the sections the A3 should contain.



A concept should answer the question "How can we have a meaningful conversation with development?" Naturally, this is a conversation between two parties. So both business and development need to meet and agree on what

content is necessary to have a meaningful conversation. Neither business nor development can come up with the content of a concept on their own. It has to be agreed upon by both parties.

As you can see in the following figure, the concept is divided into sections. Each section defines a question to be answered. Context matters, so business and development will find different sections relevant to their needs. Everyone has to agree on what sections the concept should contain. For suggestions on sections, see Mandatory and Other sections below.



Your only constraint is that the concept has to fit on an A3. So you have to prioritize and think carefully about what information really matters.

Step 2: Try it out!

Do a trial run on a product idea. See whether it works and whether it brings value. If the answer is yes, make the necessary adjustments in the live run.

Step 3: Simplify existing procedures.

If you have an existing method to document and communicate new product ideas during the prestudy phase before the decision point on whether to implement the product idea, make sure to simplify the existing procedures You should aim to replace them with the concept A3 and direct communication.

The point is to make any documentation in this phase so simple that anyone can complete it. If you can shrink your prestudy into something as small as an A3, you'll stay honest because you aren't pretending to have more information than you really have at this early stage. It would be wiser to focus your energy on generating feedback on usefulness (will someone really buy this?) and feasibility (if they would, can we do it?), rather than writing lengthy requirement documents at this early stage.

The following are examples of documents that can be simplified into concepts during prestudy:

- Market size estimation
- Business case ROI
- Requirement specification (high level)
- Overall architecture

Step 4: Make it a standard procedure.

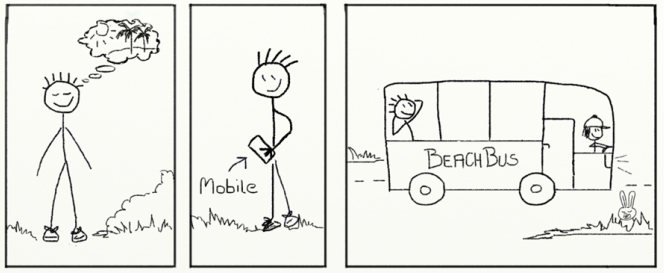
If you find concepts useful and wish to secure a high-quality conversation every time, then make concepts a standard procedure. That means you expect stakeholders to prepare a concept (more importantly, to answer the questions defined by the concept sections) before entering a conversation with developers.

Concept Layout

As mentioned earlier, the sections to include vary depending on what business and development ask for. Following are some suggestions of sections, and we note which ones should be considered mandatory.

The Mandatory Sections

The first section we want to look at is Impact, which should be considered mandatory. Impact describes the outcome we expect from the customer. It should answer the question: "What gets better in the customer's world once they start using it?" I typically draw this as a comic strip to show an example of how the world would be better if the product existed. By creating a comic strip, I provide an example that helps communicate context and understanding of the user environment in which the feature is expected to be used.



"Carl the student dreams of going to the beach"

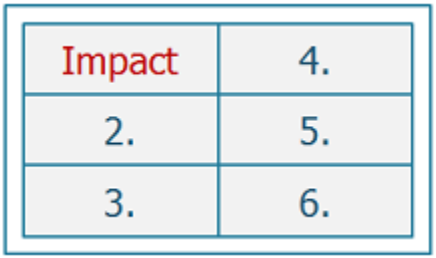
"By positioning Carl we can suggest the best commute to the nearest beach"

"Carl on his way 20 min later"

Why is it important to give an example? As we see in this strip, by describing the impact we want (Carl wants to go to the beach), we open up more options for how to solve the problem. We allow our developers to suggest better solutions (why not inform his peers so he can hitch a ride?) than what we might have originally considered.

> A requirement is the first step of a solution.
>
> ➤ *Gojko Adzic*

As the following figure shows, I typically place Impact in the top-left section of the concept.

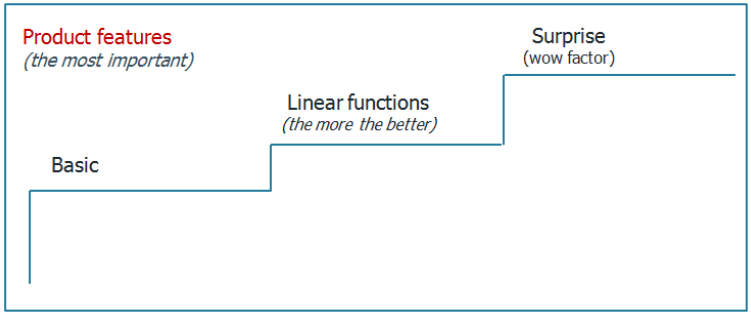


Another mandatory section is Tradeoffs, to decentralize decision-making. By clarifying tradeoffs, we decrease coordination overhead without sacrificing the intent and value of the overall idea. This section answers the question: "What unique value should not be traded away, and what can be traded if we have to overcome a prospective constraint?" As the following figure shows, I typically keep the Tradeoffs section at the bottom right of the concept.

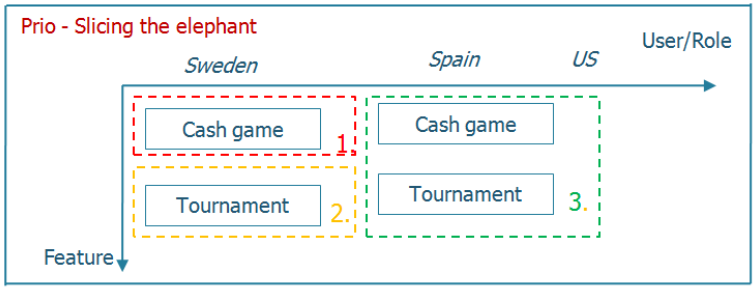
My preferred way to visualize this section is by using a set of stairs based on the Kano model, as shown in the following figure. Kano looks at features from a customer value perspective and makes it easy to visualize tradeoffs and no tradeoffs. The model divides features into:

- Basic: These are the features that need to be there for the product to be viable in the market at all.

- Linear: For these features, the more the better. For example, the faster download speed a customer gets, the more the customer will value his or her smartphone.
- Exciters: These are features that customers didn't know they needed until they saw them.



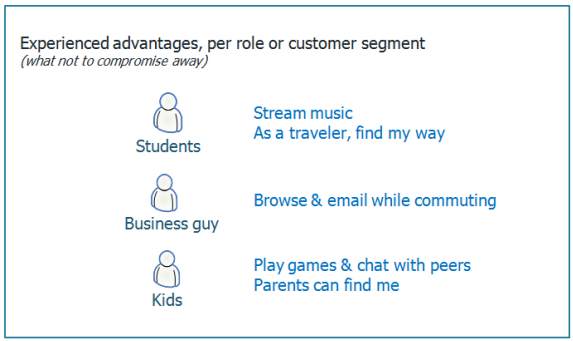
Another way to visualize priorities is by features per user group/market. In this case, we visualize tradeoffs by features and user segments and roles. The number shows the order of priority. This style is useful in larger projects and third-party software implementation projects.



Other Sections That Could Be Helpful

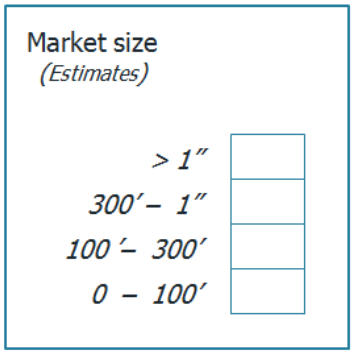
These sections are not mandatory, and business and development teams should decide which of these make sense for their purposes and how they should be used. There are many other possibilities, and this list should be treated as a starting point, not as a final list. Feel free to add what's useful for you—these are the ones I've found useful in the past.

I set aside a section for User Groups, to list different groups and the key advantages for each.

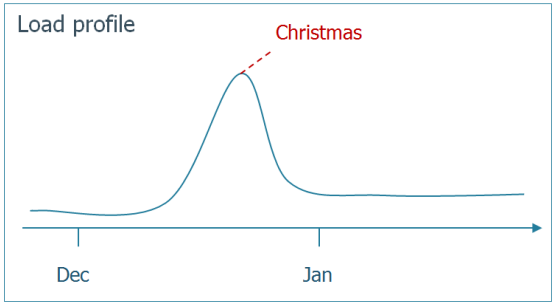


The section on Market Reason clarifies the competitive reason for pursuing this change or feature. You should be able to say why this needs to be done right now.

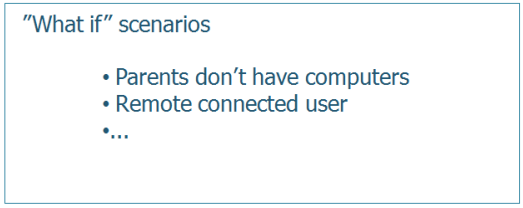
If you have an idea of the potential market size that would be served by this change, it would be helpful to include a Market Size section as part of the concept.



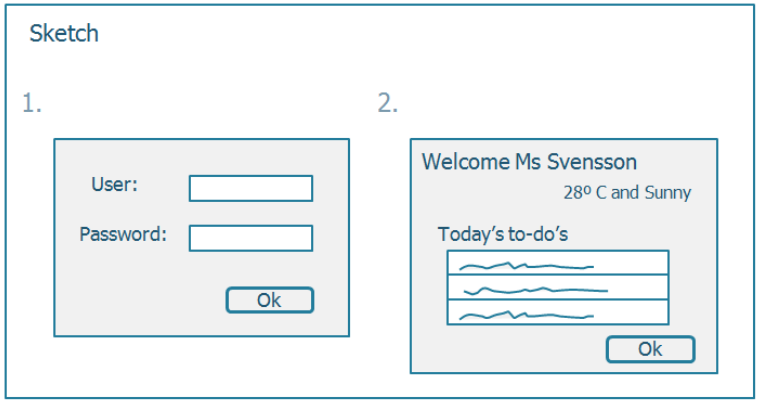
The Performance Load Profile section identifies any load patterns or peak activity that may result from this change or may be affected by this change.



The What-If Scenarios section is much more open ended and lets you clarify situations to be aware of in the user environment beyond the actual product. In this section, you can also put information about how to prepare for these scenarios.

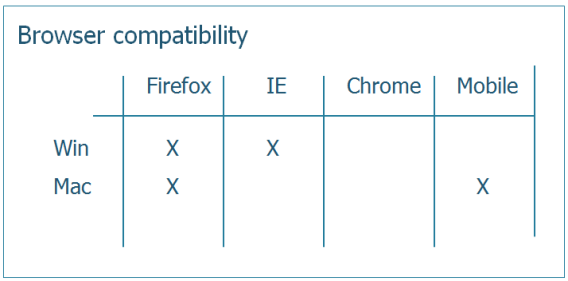


A picture is worth a thousand words. You can use the Sketch section to show the main interaction steps for the concept. You can show how you expect the scenario to look.



Collecting feedback lets you know what users and customers think about the way the idea is implemented. The Feedback section should list ideas on how you will collect feedback and from whom you can learn whether this idea actually delivers value.

Depending on the idea, you may need to think about browser compatibility. The Browser Compatibility section will list which browsers users have in their environment and thus which browsers need to be supported.



That's pretty much it about concepts. Your turn now—give it a try!

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, Raleigh, NC, and 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, Reading, 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

D

E

F

G

H

I

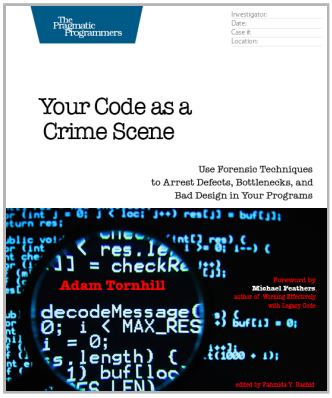
S

T

Be Agile

Don't just "do" agile; you want to *be* agile. We'll show you how, for new code and old.

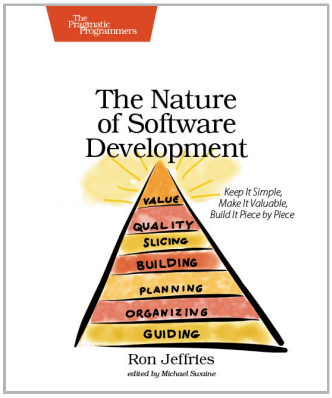
Your Code as a Crime Scene



Jack the Ripper and legacy codebases have more in common than you'd think. Inspired by forensic psychology methods, this book teaches you strategies to predict the future of your codebase, assess refactoring direction, and understand how your team influences the design. With its unique blend of forensic psychology and code analysis, this book arms you with the strategies you need, no matter what programming language you use.

Adam Tornhill
(218 pages) ISBN: 9781680500387. $36
https://pragprog.com/book/atcrime

The Nature of Software Development



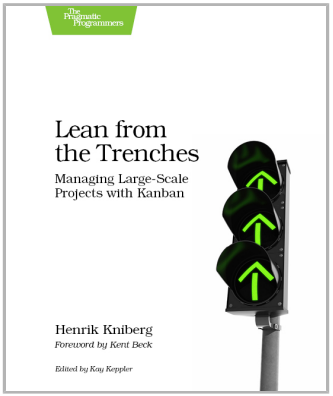
You need to get value from your software project. You need it "free, now, and perfect." We can't get you there, but we can help you get to "cheaper, sooner, and better." This book leads you from the desire for value down to the specific activities that help good Agile projects deliver better software sooner, and at a lower cost. Using simple sketches and a few words, the author invites you to follow his path of learning and understanding from a half century of software development and from his engagement with Agile methods from their very beginning.

Ron Jeffries
(178 pages) ISBN: 9781941222379. $24
https://pragprog.com/book/rjnsd

More Lean and Agile

Learn how to be more effective in your software engineering practices.

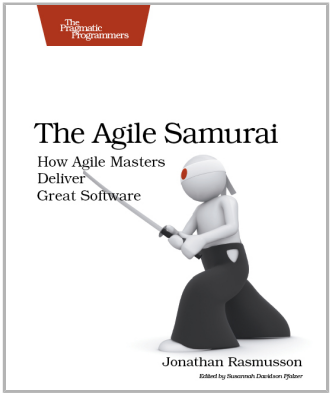
Lean from the Trenches



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.

This title is also available as an audio book.

Jonathan Rasmusson
(264 pages) ISBN: 9781934356586. $34.95
https://pragprog.com/book/jtrap

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for 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/mskanban
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
https://pragprog.com/updates
Be notified when updates and new books become available.

Join the Community
https://pragprog.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

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. It's available for purchase at our store: *https://pragprog.com/book/mskanban*

Contact Us

Online Orders:	*https://pragprog.com/catalog*
Customer Service:	*support@pragprog.com*
International Rights:	*translations@pragprog.com*
Academic Use:	*academic@pragprog.com*
Write for Us:	*http://write-for-us.pragprog.com*
Or Call:	+1 800-699-7764