The Pragmatic Programmers

Real-World Kanban

Second Edition

Do Less, Accomplish More with Lean Thinking

Mattias Skarin
Foreword by Henrik Kniberg
edited by Michael Swaine

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

CHAPTER 2

Using Kanban in the Enterprise

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.

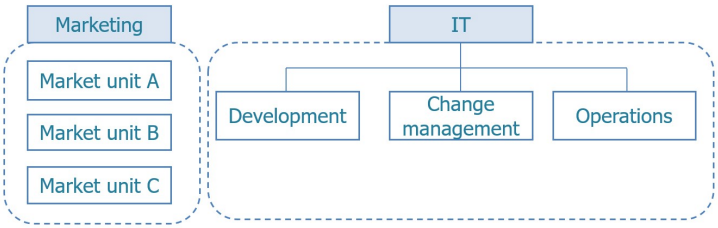
Very few companies start off improvements with a clean slate. They carry legacies—people, technology, roles, culture, market share, and processes. The legacy proves the company was successful at some point, but now results in heavy processes and structures, which slow down productivity and make things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

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

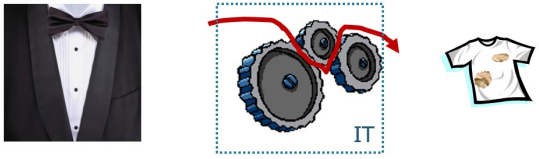
As you can see in the following diagram, the marketing department at Company H has three units, each targeting a specific market segment. IT is also split into three units: development, change management, and operations.



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 16.)

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.

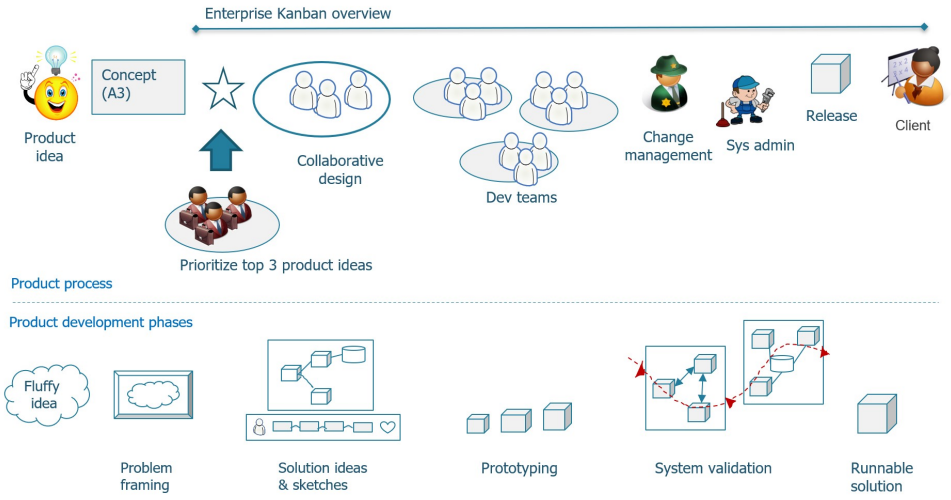
Reintroducing Sprint Planning

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.

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!

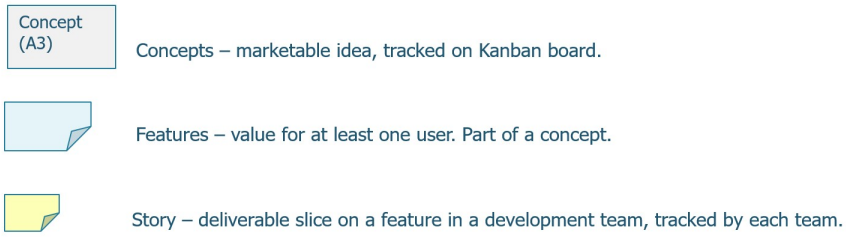
like you've already decided on the solution" a developer said during one such meeting.

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

Work with the Kanban Board

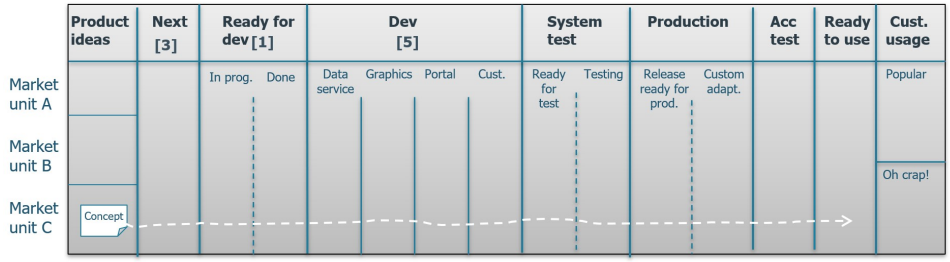
Now that we are familiar with the overall workflow, let's see how it translates to the Enterprise Kanban board. As we saw earlier, a product owner writes down a product idea, a concept. This concept is then prioritized by the head of each marketing department before it is placed on the Kanban board.

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.



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 9, 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.

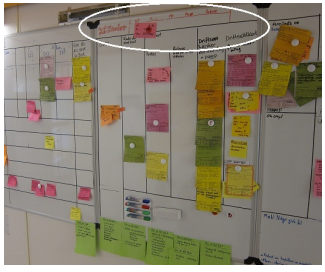
Ready to Use

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 15.)

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

Match Estimate Effort with the Decision

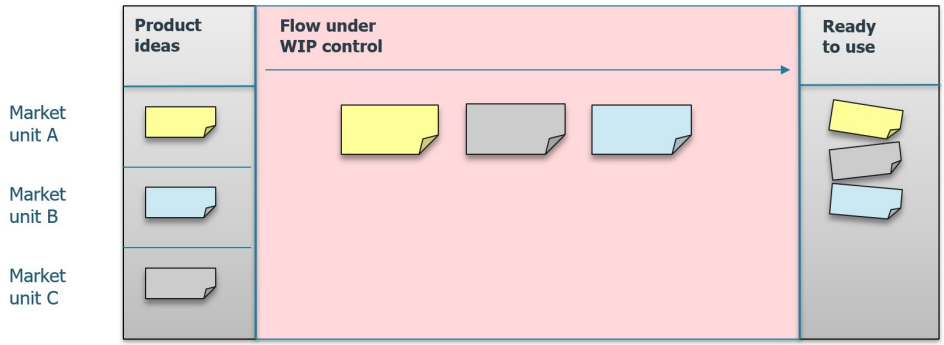
An all-too-common mistake I see product development departments make is determining cost and time estimates on too fine a granularity. There is nothing wrong with making ROI calls, but the time invested in making them should be linked to the decision you are trying to make.

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:

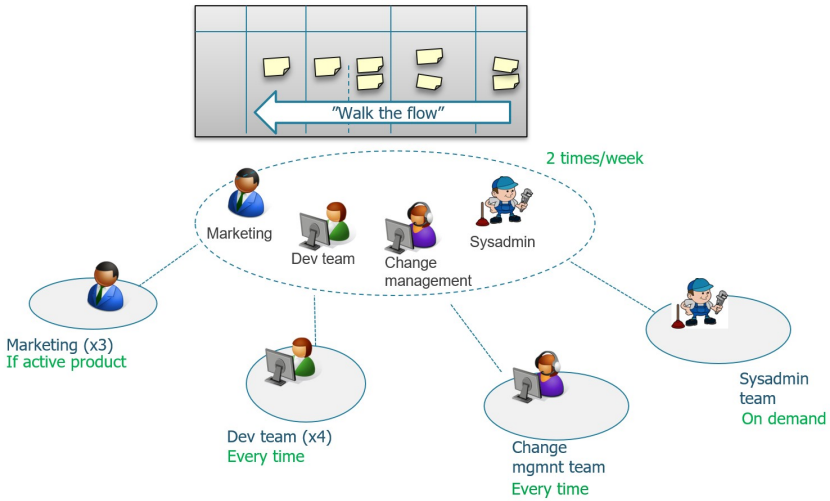
- Fifty percent of work should be product idea oriented (taken from the Kanban board).
- Twenty percent would be improvements (if there were none from the enterprise board—this was left to the team to decide).
- Twenty percent would be bug fixing.
- Ten percent would be quick fixes, answering questions,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

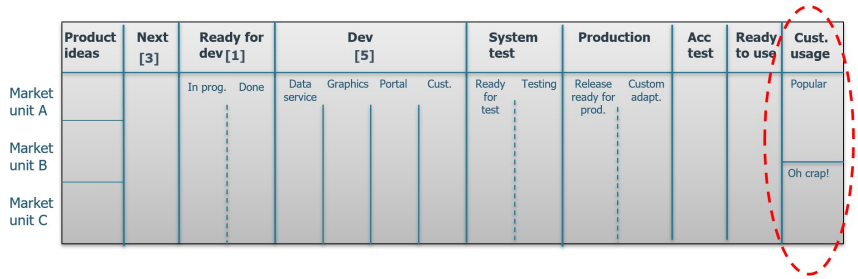
How We Continuously Improved

Take a look at the workflow again in How the Process Worked, on page 8. 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

Continuous Feedback

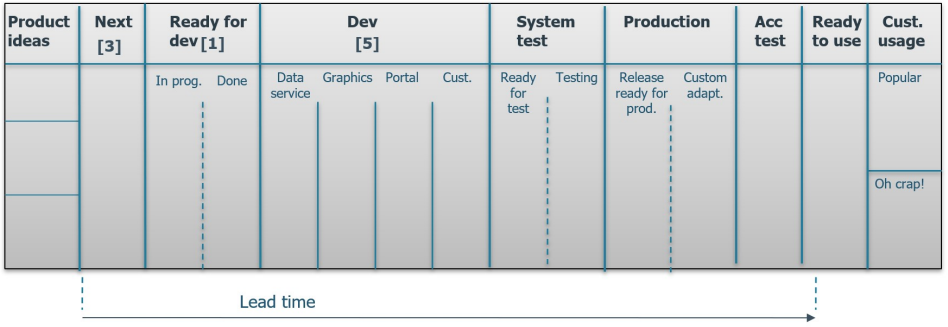
your first clue that something will go wrong. Every new product needs at least one revision before getting it right—this is your key to corrective action. The art of slicing (turning big things into small) and continuous validation of working software are two key components that can help you get it right.

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 22.

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

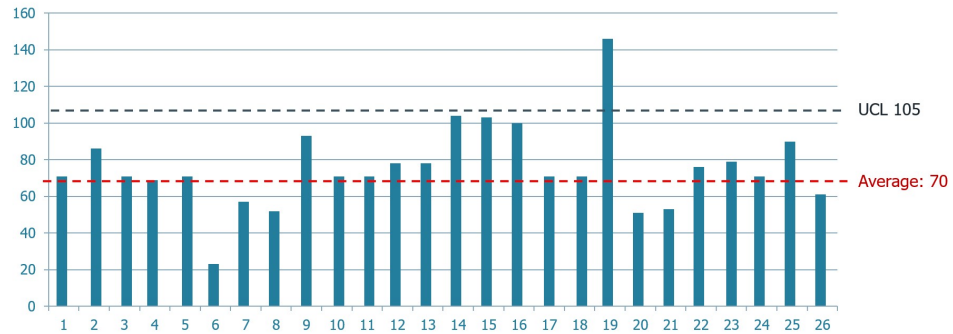
expected product features to work the first time they tested them. This helped emphasize that we were serious about the quality-first mindset.

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.

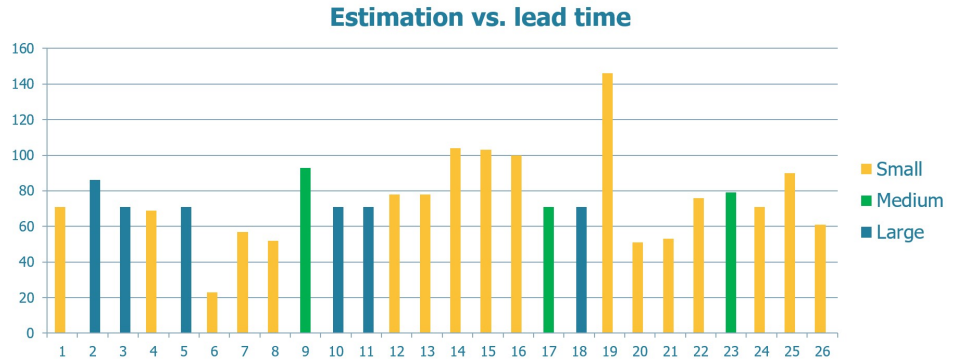


This helped marketing manage client expectations and know when to begin preparations for a new product if they wanted to hit a certain timeframe or season.

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?



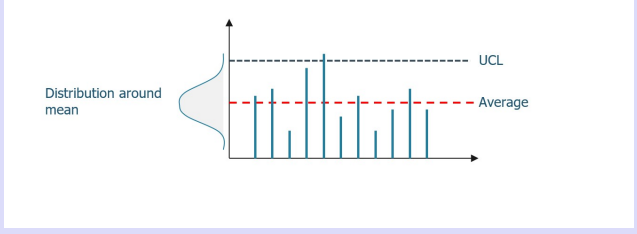
In our case, the surprising truth was a resounding "no!" Judging by the data in the chart, we can argue that there was really only one estimation bucket (or two, depending on whether you see the longest data point as a random event).

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), 68 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.

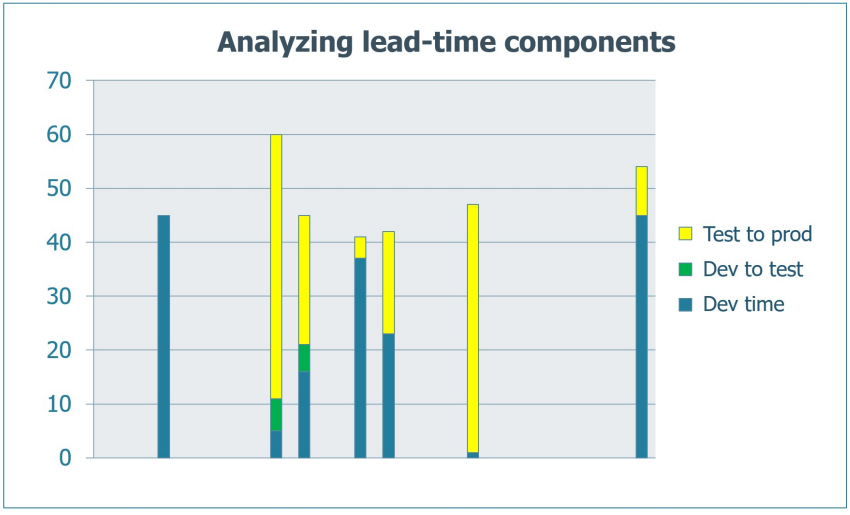
There are statistical ways to calculate the UCL, which I won't get into here. And there are hacks to estimate your UCL. Make a chart like the one above, with one bar for each lead-time observation. Draw a line across the top of the bars, skimming above the majority of them but cutting across one or two.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

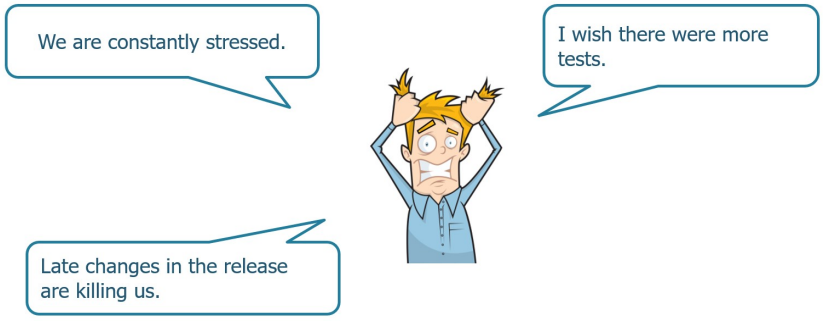
How We Located Our First Improvement Opportunity

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

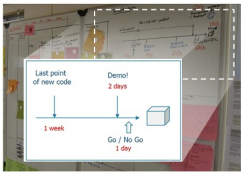
We had several examples of late changes pushed in very late during the release cycle, which made it hard to complete our system testing. We backtracked through the quality efforts required and concluded that this usually happens one week before the release. That was the time required to complete our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

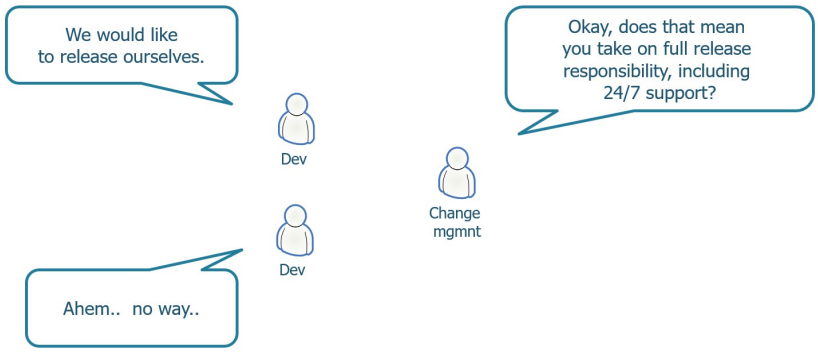
As we got more data, we saw that we could shave off a big chunk of the lead time if teams could release themselves; more precisely, if they could exempt themselves from the monthly release window. There are some advantages to this, such as the fact that we would virtually eliminate waiting time for system

testing. Smaller releases mean that it would be easier to identify causes for quality problems.

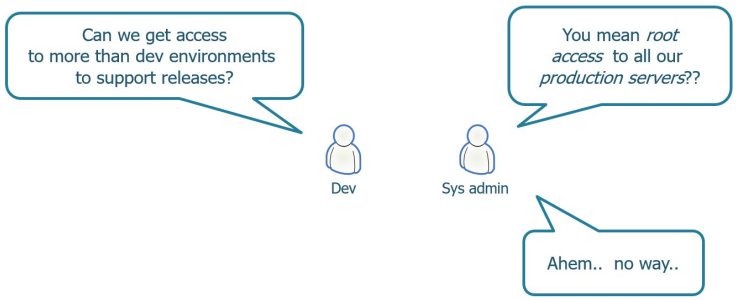
It would also remove the delay aggregation effect—such as what happens when a one-day delay near the release window turns into a four-week delay because of the monthly release window. If teams could release themselves and step out of the monthly release window, a one-day delay would be just a one-day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

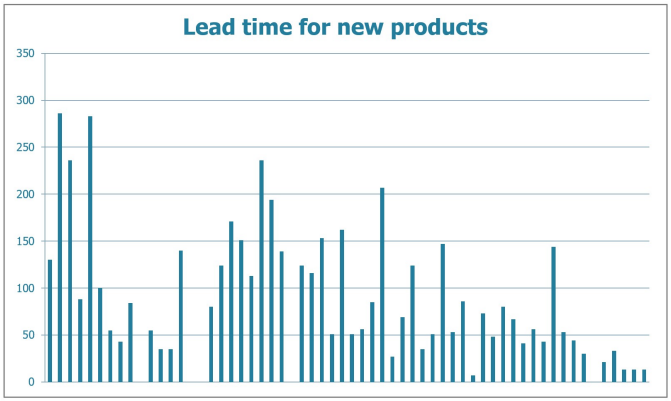
We also differentiated risk profiles. We decided it was okay to release at will for systems with few dependencies and good test coverage. We continued to release under the normal release window for systems with many dependencies and low test coverage.

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.

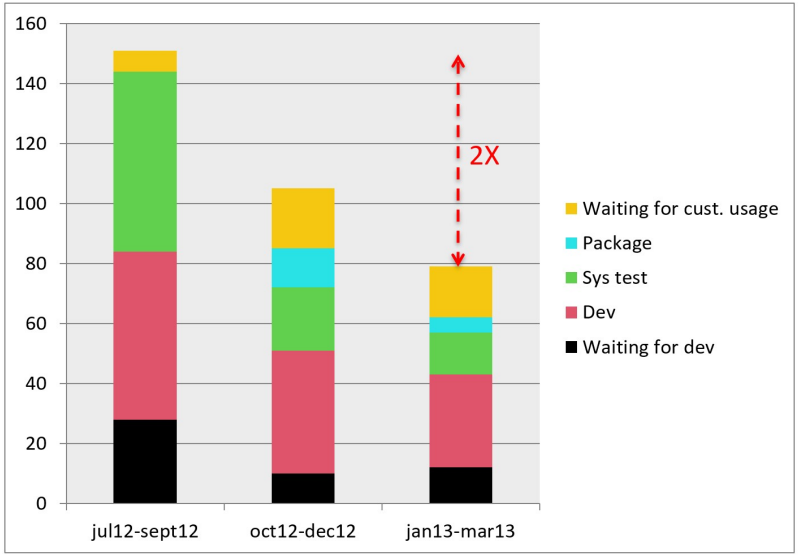
We moved through each step one at a time. It might not seem like a big deal that we finally got to a point where the development team was able to release outside the normal release window, but this would have been inconceivable two years prior.

How Lead Time Improved

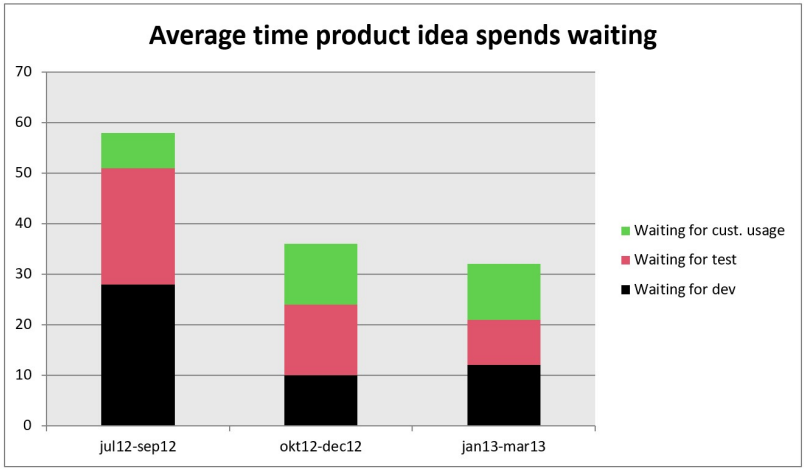
Did we improve? That's the most important question. All these changes don't matter if we don't have improvements we can point to. Let's look at some data, starting with lead time for released products.



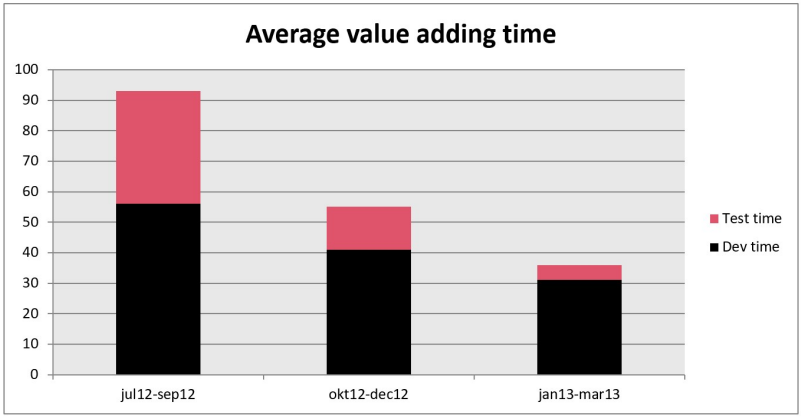
As you can see in the graph, lead time is trending downward over time, so efforts to reduce our time to market seemed to be paying off. You can get a better feel for this in the following graph, which looks at lead time for released products by quarter.



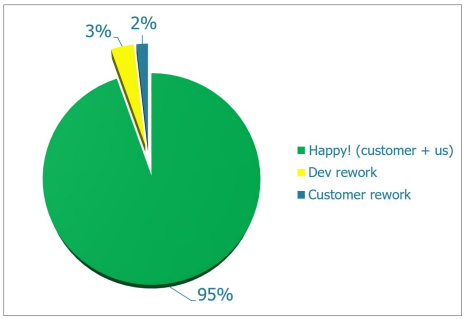
Products released through Q1 2013 got shipped roughly two times faster than in Q3 2012. This was great news, but we needed to know where our improvements came from. The following graph breaks down lead time into *waiting time* and *value-adding time*.



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).

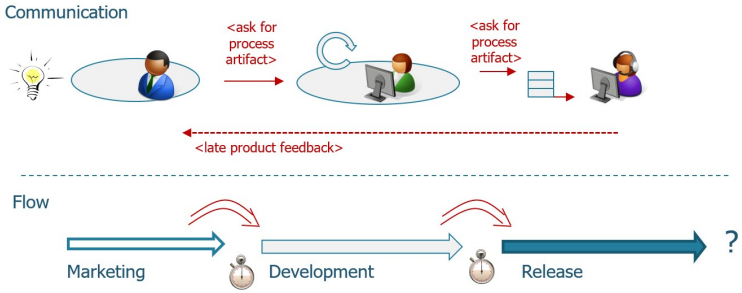
Have a Feedback Loop

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.

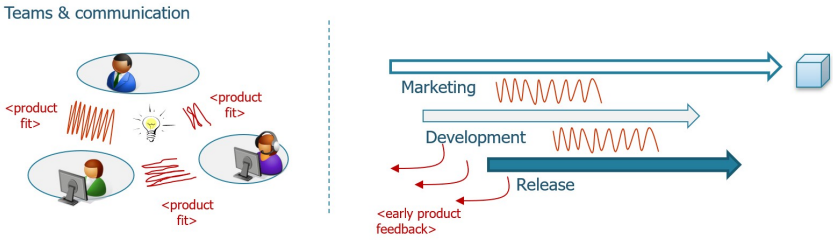
Comparing Now and Before

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.

Next steps

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