Enterprise kanban – a case study of improving the full value chain using Lean thinking

Executive summary

Very few companies start off improvements from a clean slate. They carry legacy: people, technology, roles, culture, market shares, processes. A simple proof they have been successful at some point. So how do we improve a traditional company? How do you get to a high trust culture? I find this question interesting. Refining it into a question:

“If you are a company that has been successful at some stage: you have an existing market share, you have heritage in both systems and people – then how far can we get by improving flow, step-by-step and adding skills before altering the organizational structures becomes necessary?”

This is our learning’s from improving the full value chain at a traditional company using Enterprise Kanban.

What we found

We were able to reduce lead times by half over a period of 1,5 years. For released products, 95% were reported value adding and useful.

![Figure 1 Lead time for released product ideas aggregated per quarter.](image)
**We use a minimum overhead**

We work without traditional product owners or product managers. We don’t have a project office. We rely on self organized teams, Enterprise Kanban, collaborative design, Concepts and cooperation over function borders to make it happen.

**A little bit of background**

The company sells and produces weather services. It has been around for 100 years and it might surprise you but it was actually one of the early pioneers in computing. The flipside of this is the company hefty tech stack, dating back to the 70’s. The company currently supports and runs 80+ systems.

The company employs 650 people in total, with roughly 100 of them being involved in new product development. Products are sold business to business.

They are organized into two main departments: Marketing and IT. Marketing is split into three units, each targeting a specific market segment.

**IT** is divided into three functions:

- Development (software development teams, loosely organized by systems)
- Change management (pushes changes to staging & production, responsible for system tests for releases)
- Operations (performs infrastructure work and live monitoring)
Why get started?

**Faster time to market to stay competitive**

Openness of data and deregulations has created new competitors. Being new they don’t carry technological or organization legacy. Thus they are fast moving, aggressive and provide stiff competition.

Our key challenge was to improve our ability to ship products faster, crucial to stay as a competitive partner on the market.

**Experience: Big projects that never seem to finish**

Never happened to you right? The company had just stopped an earlier project, aimed at renewing the product platform, which had been running well over 1.5 year and still was far from finished. The bulk of current development efforts aimed at solving this once and for all using new technology, better architecture and agile teams.

“I requested a suit and all I got was a lousy t-shirt”

This is an actual quote from a marketing manager describing his view on product development. It was hard for marketing and development to agree on the right level of communication. Either the make or break information such as what was *unique* in a product idea wasn’t explicitly communicated– or - if it was, in order to make the deadlines, teams would abandon these uniqueness in favor of shipping the product in time. Either way, the upside of the product got lost.

**Where’s my product idea?**

It might seem odd, but no one really knew the exact state of current products ideas. These product ideas could exist partially in several Scrum teams product backlogs and also could be in different stage 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 for marketing to see what stage the product idea was in.

Another benefit of our enterprise kanban board was to provide a shared language for all functions to discuss progress, and enable us to get a shared focus when needed.
Selecting the right language in communication between business and development is tricky. For example: should we choose a business friendly visionary language (“here are the three key areas our product needs to excel at: usability, reliability, pricing..”) and let the developers iron out the details? Or, should we opt for a development friendly language (“the acceptance criteria for feature ‘making a call’ is.. “)? The degree of product knowledge in the development teams and the degree of slicing skill in business sets this level. We used Concepts to help bridge this gap allowing the level of details to adapt to the knowledge in the business/team relation.

**Wouldn’t a traditional project office solve the problem?**

A traditional way to overcome knowing the state of things is to add a middle man to bridge the gap between customer and IT – product owners or project management. But by doing so we have also insert two new handovers, increasing the chance the important information get’s lost or distorted. This can be seen as a consequence of inserting roles to handle details where as the original problem was finding the right level of language that enables business and developers develop a shared understanding of what to develop.

Another challenge with traditional project management is that communication of progress is often made towards plan (using gated milestones). This is not the same as actual progress of the product. This can produce a misleading feeling of wellbeing and being on track, only to get nasty surprises towards the end - pushing the delivery date by a half a year or so. This is what agile teams long have known and helps us address.

**It’s about selling the product too**

A product becomes little worth unless we find ways to tell the market about its existence, sell it, integrate it, train customers in its use and support it. Without integrating with the steps outside software we won’t bring in the money. So we need to look beyond development, accept existence of functions (or become magnificent multitaskers) and find ways to interact over the full value stream.
“I feel like I’m just a small cog in the machinery”

Developers felt like they merely was a small cog in the machinery and lacked an overall picture of what they were creating. We wanted to raise teams ability to take higher responsibility on the overall results (product idea success).

How we got going

Where to put the board

The kanban board ties together four functions: marketing, development, change management and operations. It visualizes flow from product idea creation to customer use. We decided to put the board in a corridor outside the development teams, through which most of the involved functions pass once in a while.

After putting up the board the next step was to fill it with ongoing and upcoming product ideas. Mapping current sprint items to ongoing product ideas was a fun challenge that took some conversations to get right. But by the end of the day we got them up.
Goodbye sprints, welcome flow

We wanted to refocus teams on flow, rather than sprints. Why?

- To shift focus on finishing product ideas, over completing team increments
- To allow communication across functions to circle on product ideas, less on process artifacts.
- To enable us to work on product ideas until done with quality, not shipped because sprint has ended
- To find bottlenecks and eliminate bottlenecks in end to end flow
- To eliminate wait time

At first development teams were cautious about this change, in their view sprints worked. But they were also keen to get a better view on what they were working on (less a ‘cog in the machinery’) plus they were curious and so they agreed to give it a try.

Removing sprints - what we learned

After a couple of months a few development teams reintroduced “light” sprint planning. They lacked a team overview of what was being worked on, and wanted to use combined skills when splitting complex work. It was ‘light sprint planning’ since it excluded estimating how much they could fit into sprints (we worked with continuous flow).

How we approached product ownership

Traditionally, product management or product owners – are responsible for product decisions. We took a slightly different approach. We wanted the passionate people behind the idea to run with it, regardless of role. But in order to do so we requested two things:

1. You run with the idea all the way to working client.
2. “You want it – you make it happen” – there is no handover

We called this approach Concepts. Concepts helped us to:

- Keep the integrity of the original idea through all phases of development
- Get a feedback loop from client use - post release
- Make sure business owners arrived prepared to the conversation with developers
- Share the big picture across multiple teams
- Stop ideas early which no one really cared enough for
Decentralize risk mitigation. Concepts allowed us to empower any team or downstream function to make tradeoff decisions without asking for permission.

To learn more see: Introduction to Concepts.

**How we handled the “overall experience”**

It is fair to say that the overall experience was handled from a market segment perspective. Each market department maintained their product portfolio and knew what product ideas that were under development. If they discovered that an effort was required to improve the product portfolio targeting this market segment, they would either:

- Insert a new concept (or)
- Ask an existing concept owner to make adjustments necessary to improve the portfolio product experience.

An example of such efforts was performance improvements.

**Learning to prepare good inflow**

In the beginning we kept candidates for new product ideas on a wall next to the kanban board. Since we used Concepts, each product idea was represented by an A3.

![Figure 3 Our wall with candidates for new products](image)

The first time I reviewed them, I noticed 40% did not contain the specified information we had agreed necessary to engage in a conversation with development. For example, **impact** would frequently be missing. To fix this we added a policy where...
we requested that the team leads screened incoming Concepts before they entered prioritization and remind the creator of any missing information. While this was partly expected since we were still early in the learning curve, one can’t stop wonder what would happen if we had developed and released these product ideas.

How we approached ROI and budgeting

A common scenario is to do a return on investment calculation (ROI) for new development projects. Costs are generally estimated by asking each function to predict number of man hours involved. This is used to decide if this is a profitable investment and sometimes to figure out how much of the IT budget that this project will consume (and by whom).

We have to make calls on what to develop. So we do a value vs effort judgment one way or the other. The problem happens when our effort is largely focused on estimating cost rather than value. The value of the product idea normally carries higher uncertainty than the cost. Therefore, spending large amounts of effort estimating the cost side of the equation is not well invested effort (you are addressing the wrong uncertainty).

Try a simple thought experiment: how many customers will buy your product after it’s been released? Estimate the range for the value comparing worst case with best. Then estimate the range of costs (hint: Standish group estimates overrun in costs in software projects to be 240%). Compare the ranges and see which one that carries the most uncertainty. Now ask yourself how much of your efforts are directed to each.

![amount_of_uncertainty](image)

Figure 4 The uncertainty imbalance. The biggest uncertainty is on the value side - will the product fly? Yet we invest the most of our effort during early product development in estimating the effort/cost.
We wanted to shift emphasis to reducing value uncertainty. So we simplified our return on investment calls (what to develop) using a set of assumptions:

- Uncertainty of value is best discovered by shipping the product idea and trying it on the market.
- The cost, of which the headcount is a main component, remains fairly stable over time. Changes do happen, but they are regularly a rare event.
- Time through the system = effort consumed. Use lead time for product idea to learn how much effort it has consumed.
- It is smarter to validate effort consumption throughout or post development than pre development.
- Ensuring that each marketing department get’s their fair share of the development effort, matters. This can be guaranteed using the right inflow and WIP limits.

Funding of IT was solved by letting each marketing department fund IT with an equal amount (1/3 each). In return, they would be guaranteed to get every third product idea. In this way, the important decision each marketing department needed to do was figure out what the next product idea most likely to succeed on the market.

![Diagram](image)

Figure 5 A fixed WIP combined with the decision rule of “each marketing department got every third product idea made sure each department got the portion of development effort they paid for.

The default decision rule of “every third product idea” could be overruled, if marketing department heads agreed. This would typically happen if they recognized that a certain product idea or improvement gained the company as a whole. Heads of marketing met in front of the board every 14th day to review priorities and make such calls if needed.

One example of a prioritization call that overruled “every third” was to pull in a performance improvement that gained all departments.
Enterprise kanban - walking the board

A brief explanation of each step

<table>
<thead>
<tr>
<th>Product ideas</th>
<th>Each marketing department is responsible for keeping two product ideas prepared here. For a product idea to exist on the board, it had to have a prepared Concept.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next</td>
<td>These are the next three product ideas the company has decided to pull in. This is also where WIP begins and lead time measurements start. From this moment the marketing department cannot insert new features or make major changes to the feature scope. They do have the right to cancel the product idea all together, in this case it moves to “oh crap” section at the end of the board.</td>
</tr>
<tr>
<td></td>
<td>The manager of each marketing department met up in front of the board every 14:th day to review prioritization. After a while our head of development also joined this conversation. This proved to be an ample opportunity to discuss and agree why an investment in technical debt would be beneficial now or later.</td>
</tr>
<tr>
<td></td>
<td>Not all prioritization decision needs to be synced between the heads of marketing. In our case each marketing department funds IT with 1/3rd of its budget. Given this, our default decision rule was every third product idea would be pulled from your marketing department. This default rule could be overruled if all heads of marketing agreed. This would typically happen for product ideas where the company as a whole benefited or key technical debt.</td>
</tr>
<tr>
<td>Ready for dev</td>
<td>Solution options under creation. At least one representative from each team participates. See more under “the collaborative design session”.</td>
</tr>
<tr>
<td>Dev</td>
<td>Product idea under development. Each column indicates roughly which team that is moving it forward right now. Before the product idea moves to system test, teams and concept owner must agree that the product and it’s features represents a sellable product.</td>
</tr>
<tr>
<td>System test</td>
<td>Basic verification product works from a system perspective Integration and deployment on production like platform, maintainability, data and</td>
</tr>
</tbody>
</table>
stability over time verified.

<table>
<thead>
<tr>
<th>Production</th>
<th>The release is moved into production. Customer specific branding and configurations added, if necessary.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc. test</td>
<td>Validation by concept owner of final experience.</td>
</tr>
<tr>
<td>Ready to use</td>
<td>Ready to be taken into use by customer.</td>
</tr>
<tr>
<td>Customer usage</td>
<td>Feedback from customer. If they liked it and uses it (=“popular”) if it didn’t work out (=“oh crap!”)</td>
</tr>
</tbody>
</table>

**Kanban standup - sharing progress and addressing flow blockers**

We decided on using a regular meeting cadence in front of the board, 2 times a week we would gather key stakeholders and have a 15 minute standup. The purpose was to allow stakeholders to get an overview on the state the product ideas and to address impediments blocking progress.

**Who comes?**

At first, we asked for a (minimum) representation by each function at the standup. That made up three 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 would get pulled in “on need” basis. Our facilitator at these
meetings and the owner of the enterprise kanban board was our head of engineering.

Our agenda:

1. Walk the flow. Find out if there is any blocker preventing progress. We walked this from the back of the board forward.
2. If a blocker is found: Ask who are the right people to address it and assign one responsible for each. Avoid discussing the problem at the standup, do this right after.
3. Before ending the meeting: Recap important points (for example who owns each blocking event)
4. Officially end the meeting (“Thank you everyone – we are done for today”)

It is key to keep these meeting short. We would have 10-20 people in front of the board from different functions at these meetings. We learned to arrive prepared and study the board in advance instead of tripping over at the typical “oh, what was this about again..” with people patiently waiting. I believe we can say it worked, since we still use this twice a week standup.

**Kanban standup –what we learned**

Operations remarked after a while that there rarely was an item on the board that required their input, so we simplified attendance to: “*If you have something of interest on the board you come to the standup.*” That worked better. Using this approach Concepts owners (marketing people who had active product ideas under development) would always be there, one representative from each development team (if they were working on an active concept) would be there. We would pull in specialized resources such as sysadmins or outside teams on need basis.

**The importance of the overall blocker section**

We learned early on, that a few key impediments could not be tied directly to single product ideas. Some spanned many. To ensure we acted on them, we added a section on top of the board, where each development team could signal if they were being impeded by some factor.
While it may seem trivial, it 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 these things mattered.

It was frequently used in early during our kanban implementation, but as we solved a couple of the key blocking issues, it got less frequently used. Currently it is rarely used. It is still on the board mainly to ensure 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 head of marketing and head of development.

**Learning's - why weren’t these things solved before?**
You might wonder why some if 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, sometimes functions to address. Now we were able to focus across functions to address them.

**How the team decided what to work on**
A question that surfaced early was “what should we do about work the teams face but that is not on the kanban board?” We clarified to the teams that we wanted them to keep visualizing what they were working on and gave them a decision rule to organize their work around.

- 50% of work should be product idea oriented
- 20% would be improvements, (if there was none from the enterprise board this was left to team to decide)
• 20% on bug fixing
• 10% on quick fixes, answering questions etc.

Each team was given the mandate to make the call if to pick up work or not when approached by external parties as long they kept these rough guidelines.

**How we kept track of spent effort**

On top of the kanban board we kept a small section that was updated and reviewed at the monthly retro. This section was updated by the teams in the presence of the IT managers. Each team was asked to give their picture of their effort allocation by manually changing the size of the columns for each category. If a big diversion was discovered this would lead to questions by the IT manager what had caused this and potential action points.

These bars was visualized on top of the kanban board, see the highlighted area below.

This proved to be a remarkably simple mechanism allowing us to track extraordinary events as well if teams had been pushed in the wrong direction for political or
personal reasons. This mechanism replaced time reporting as a tool to learn where “team had spent time”.

### How we approached collaborative design

When a new product idea comes to light it’s important to look at it from series of perspectives. How it affects architectural constraints, how architecture might affect the integrity of the idea, what type what solutions options exist, what parts need to fit together before we have something valuable, what timeframe should the design be optimized for.

We set up collaborative design to achieve three things

- **Leverage of brainpower.** If we got multiple minds to look at a problem we would get multiple perspectives quickly. No need to wait for an iteration or two to find out if it’s doable or not.

- **Avoid the feeling that teams where only “a small cog in the machinery”**. Since one member of each team participated they would bring back an understanding of the problem addressed plus the reasoning behind design decisions to their team.

- **Get a ‘creative height’**. Avoid turning breakdown a lame exercise of fitting the product idea into the existing architecture. Our goal was to always deliver two solutions to any problem.

### The workflow

A product idea is written down by the owner of the idea as a Concept. This then (hopefully...) get prioritized by the head of each marketing department. It is then pulled into collaborative design which happens on demand.
Who attends?

Calling the meeting and running the show would be done by a facilitator. He or she was generally one of the developments team’s scrum masters who received specific coaching for the job. The facilitator would call the meeting and make sure one developer from each team participated. If required, specialist resources would be pulled in to this session, for example if integration was required with outside teams.

Collaborative design session followed the agenda:

1. **Paint the picture** – the concept owner describes the idea, walking through the concept
2. **Discover** – Carve out 2-4 solution ideas
3. **Explore** – Dive deeper down each solution idea
4. **Select** – Select two options to move forward with

It’s the facilitator’s role to move the group between these modes, selecting the balance between diving in to detail and exploring options.

**Good facilitation is key**

The job of the facilitator is to:

- Make sure multiple solution options were explored
- Bring back discussions deterring into detail
- Explore ideas thrown out unintentionally but lost during conversations,
- Provoke the team if necessary to stimulate thinking from other angles. “So this is a valid solution, but what would the dirt simple solution look like?”

One of the most important tasks for the facilitator is to *transcribe the discussion*. Ideas are many times thrown out and unintentionally lost. Transcribing enables the participants to recap earlier conversations and solution options.
Collaborative design starts with picking a balanced team
If the crowd is too homogenous, few ideas emerge. Facilitation starts by picking a set of personalities balancing each other. The goal is to see different angles and enable the team to build on each other’s ideas. Being part of these sessions was focused work both mentally and emotionally. Membership rotated, and it was the facilitator’s job to keep a balanced team.

Collaborative design – what we learned
In a few examples we let a senior developer and the concept owner “pair up” and walk through the idea, before the collaborative design session. That didn’t turn out the way we expected..

The participants felt that no matter how the asked, one solution was “pre decided”. One participator actually said it out loud: “why am I here to give you ideas? It seems you already have decided in the solution..”. So we learned to bring in fresh problem statements rather than half finished solutions.

How we did continuous improvement

Learning from outcome
Continuous improvement can be run in many ways. We decided the most important information came from the usage (or in worst case: none usage..) of our products. So we made this information the foundation of our improvements.

We started by visualized the outcome of our development at the end of the kanban board.
If the customer did not like the product idea it was put into the “oh crap” area. Vice versa if the customer liked it and it became frequently used (“popular”).

If a “oh crap” event occurred, we would bring together the concept owner; the teams involved and facilitator and do a root cause analysis together. This then became input to changes we needed to do.

Adding a company demo

One of key changes we did when we started Enterprise kanban was to replace sprint demo’s with a company demo, which we ran a day before release. At this event new product ideas would be demonstrated by the teams, allowing people throughout the company to see what was going out.

But we included a second step. Based on the previous release, marketing would demonstrate how products where being in use and share customer feedback and comments. This was much appreciated by development teams and gave engineers the opportunity to learn about how the products where used by the clients.

Company demo vs. sprint demo – what we learned

At the previous sprint demo development teams normally would demonstrate components of the product idea - this made conversations at these events slightly development focused. When we shifted to company demo - demonstrating working product ideas conversations shifted towards market and product use of new products. Sprint demo’s were replaced by continuous demonstrations, as concept owners continuously reviews and exchange feedback on ongoing work with development teams.

How we chose what to measure

We decided to measure two things: Lead time (including its components) and percentage of stories reaching “Customer popular” stage vs. “Oh crap” (see above).

Lead time measurement starts when the product idea enters WIP (“Next column for us) and stops when customer can use it (“Ready to use”).
Together, these gave us indicators over time if we were doing the right thing and how we were improving on making that happen (lead time).

**Act on information – don’t store it**

We complimented measurements with a number of visual indicators. Examples include blocking events, queues, age of product idea and rough estimation of where team spent their main effort (see “How we kept track of spent effort”).

These where designed to help management and teams enter conversations and take actions. The upside of acting on it “right away” is the information and chain of events is fresh in people’s minds. If you try to problem solve this a month later the chain of events can be hard to recall.

**A monthly improvement pulse in front of the kanban board**

Once a month, the manager of the IT department (our kanban board owner) pulled together one representative from each development team for a quick retrospective. We performed this standing in front of the board asking “what should be improved?”. The retro is typically very quick, take 15 minutes, and changes to the board and process are applied their and then. The agenda followed:

- Review measurements (lead time, customer usage)
- Review board (is it clear, easy to overview, useful)
- Apply change

Examples for changes introduced by the improvement pulse have been, changing the templates for the kanban cards, inserting, removing and reinserting swim lanes, refined lead time measurements.

**Monthly improvement pulse – lesson learned**

One observation we did was that team/or -major blockers rarely where discussed at these events. Why? They had already been addressed. If a team or product idea got blocked for some reason (performance issues, release issues etc), these issues had
already gotten attention and where being solved. Thus we rarely spent time discussing fixing blocking issues at our monthly improvement pulse.

### When can I get my stuff?

Since weather business is highly seasonal (doh!) 😊 - it matters for marketing to know when something will be done. The most important decision point is to know when to feed in a product idea in order to get it out before season begins.

There was a second reason why it mattered for us to know the decision points for new products. We had examples of very late changes pushed into releases and jeopardizing quality. We needed to find a way to agree between marketing, development and change management when decision points really were. Until we did, it would always be up to each person to figure this out, a complex call no single individual could successfully make on their own.

Before we had relied on story point estimations and sprints to give us this data, but the predictions had been pretty poor. It is quite natural as in our case sprints consumed a smaller portion of the total lead time.

### How we found our true capability – time to deliver

We sampled the lead time for the latest released product ideas and figured out under what ceiling a majority of observations would fall. We chose a ceiling under which roughly 95% of events would be below – often referred to as the upper control limit (UCL).

---

**Figure 8** Our first sampling of the delivery time for new product ideas. Each bar represents a delivered product idea and the vertical scale days it took to deliver it. An upper control limit of 105 means 95 out of 100 product ideas will get delivered before 105 days.
Striking an agreement of the latest point of commitment for new product ideas

This information was used to strike the agreement with marketing of the latest commitment point for new product ideas.

This allowed us to simplify responsibilities: Marketing could focus on figuring out what next product idea to pull in, 105d before it needed to roll out. We would make it our job to deliver it before that time frame. It also allowed us to clarify that no longer would it be ok to push in product ideas “from the side”, because later changes would jeopardize the quality of this and other ongoing product ideas.

But size matters! Right?
I’m frequently confronted with the argument “but we need upfront estimation, some items are bigger than others”. Indeed that is true – some items are bigger. And if you look at the chart some bars taller than others, thus taking longer time to deliver. The interesting question becomes: How does this correlate with the upfront estimation done by developers?

We do a very simple T-Shirt like bucket type sizing before development begins.

- Small = 2-3 days
- Medium = 1-3 weeks
- Large > 1 month

Below you’ll find a plot correlating the initial sizing estimates against the lead time output. Have a look; is the initial sizing a good predictor for when you can get your stuff?

In our case, the surprising truth was ‘no’. But there are factors that affect the time to ship. In our case the more dominant factors where: wait time for release, wait time to get access to specialized skills.
Judging by the data above the argument could be raised that that there’s in reality only one estimation bucket (or two, if you don’t judge see longest data point is a random event).

**Timeout – How can I find my upper control limit (UCL)**

An UCL is calculated such that a majority of events is expected to occur below its line. The UCL reflects what degree of certainty you want to get for your predictions. You can choose to be 95% certain (2σ), 67% certain (1σ), or 50% certain (aka average – not recommended).

How big certainty you choose to opt for is up to you, I generally select UCL at 2σ - under which 95% of events are expected to occur.

![UCL Diagram](image)

**Figure 9** A way to visualize UCL is to imagine a frequency distribution centered around mean. The further away from the mean we move, the fewer occurrences we expect to find. Thus UCL represents a cutoff point of the tail of the distribution.

**The crude way of finding your UCL**

There are statistical ways to calculate the upper control limit which I won’t go through here. But the dirt simple way of finding it, you can do manually. Make a chart like the one above, using one bar for each lead time observation. Then draw a line across the top of the bars skimming above the majority of them but cutting across one. The level of certainty you will hit this number is roughly (number of bars above the line) / (total number of bars).

It is a rough and crude, but it’s a good enough approximation especially if you are under time pressure, have little data and need to make a call fast.

*Please avoid* the temptation to give delivery estimates based on the average lead time. Remember the average implies 50% of your deliveries will be below this number. A pretty low hit rate..

**How we improved lead time – step by step**

The first challenge was to understand where the opportunities for improvements where. To learn that we tracked the key components of lead time (waiting for dev,
under development, waiting for system test, system testing). Our first lead time chart was the one below.

![Analyzing lead time components](image)

Figure 10 Our first view on which components consumed lead time. A typical situation in a product development – making decisions with but a few data points. It is in situations like this where expertise and context knowledge matters.

So what should we start improving? We were leaning towards “test to production” being our biggest improvement area, but as you see the data is not terribly conclusive.

This is actually a typical situation in product development, you are moving forward with rapid pace and under high degrees of uncertainty. We displayed this chart to highlights the limitation of how much information you can read out from a chart. It’s in situations like these context understanding (seeing with own eyes) and experience matters to make the right calls.

So to get context understanding, we needed to listen to the source close to the problem. This meant walking over to the change management team and to find out how they viewed the situation.
Hm. Sounds pretty much like a bottleneck right? Change management seemed like a good point to focus our improvement efforts on.

**Improving @ change management**

The change management was a team of nine people working to roll out changes in 70+ systems with technology stacks dating back 30 years in test and production. Obviously they had a lot to do. After discussing with the management team in change management we decided to attack the problem on three fronts:

- *Introduce kanban in change management* - enable them to work on the high prio stuff, gain time to get the quality right, decrease stress and enable teamwork. (see separate article “Kanban in change management” for the full story)

- *Stop doing late changes in the release* - we struck an agreement between dev and change management about when is the last time a change could be accepted – and keep it

- Engage developers in adding automated test scenarios to our system test environment – this would simplify testing and more importantly test feedback

**How we stopped doing late release changes**

We had several examples of late changes being pushed in late, during the release cycle. Obviously we lacked a common agreement between development and change management, and even between change management team members when the cutoff point for late changes really was.

We backtracked the quality efforts needed and conclude that this point was *one week before the release*. That was the time required to run the key system tests.

Because of immense time pressure, late changes and inability for a single change management agent to overview the status of the release, system tests were often only partially run in order to move the release out in time. We needed to find a way to change behavior to build quality in, instead of pushing quality out.
Here’s a typical scenario where problem solving must involve several parties to be successful. One single person/or function cannot make this happen single handed. The only way this would work was if the agreement was kept and respected by all parties - all development teams and all change agents.

To visualize this we drew a timeline at the far end of the kanban board that included the cutoff dates and we asked the change management team to update this for us for each release.

**Figure 11 The last point of change - visualizing the release timeline at the kanban board.**

**Hey - didn’t you have a process before?**  
The answer is yes! We had a very hefty and well documented process, that dictated how things *should* happen. For example, 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 later changes were doable.

Just because it a process and it is documented does not mean it describes how work *really* happens. This is part of the beauty of kanban; it describes a living process, “as is” *right now*, not as it was expected to work.

**Breaking the last barrier – letting the development teams release themselves**  
We had a long term ambition that teams should be able to release themselves, instead of waiting for the upcoming release window. If we made this happen, it would give us a number of advantages:
- Decrease average time waiting for system testing
- Smaller releases means simplifying solving quality issues
- Focus on releasing the product *when it’s right* rather than because *it’s time*
- Remove the *delay aggregation effect* - a one day delay near the release window instantaneously aggregates to 4 weeks (a full release window). If the teams could release themselves - a one day delay would stay as one day delay.

Making this change may sound simple in theory, but proved hard in practice. When we started to discuss this idea with the developers and with change management, the discussion derailed pretty quickly. Let me share two examples:

Situation #1:

```
We would like to release ourselves

Ok.. Does that mean you take on full release responsibility including 24/7 support.

Ahum.. no way..

We would like to release ourselves
```

Situation #2:

```
Can we get access to more than dev environments to support releases?

You mean root access to all our production servers??

Ahum.. no way..

Can we get access to more than dev environments to support releases?
```

Dev
Change mgmnt
Dev
Sysadmin
Sysadmin
Dismantling the problem

We found ourselves in a mental deadlock and a catch 22 situation. The way forward was to break this down to small and very concrete steps that made sense.

1. Be clear we are not trying to decrease quality. If we do this it’s to improve quality.
2. Learn to respect the late change cutoff date
3. Get a dedicated change management team member for each team
4. Free up time - allow change management team members to spend 50% of their time working alongside their development team
5. Clarify the expectations of the release work, create a release checklist
6. Find a sustainable way to give development teams access to test and production environments allowing them to prepare releases and help troubleshoot
7. Let teams make the call on when to release, outside normal release window, then execute it with support of the change management person
8. Let teams do their own release, using the release checklist

We moved through these steps one at a time. We are at point 6. currently - in October 2012 the first teams did changes outside the normal release window.
How lead time improved – crunching the data

Let’s have a look at some of the data we collected, starting with the lead time:

![Lead time end to end flow](image)

Figure 12 Lead time for new product ideas. The two data points at the end mark tech debt stories.

As you can see, lead time is trending downwards. The two data points at the end are Tech debt stories, aka improvements for keeping the pace up in the future. In our case they were generally of high complexity and carried lower priority, this we can expect lead time of tech debt stories to be longer.

**Lead time per quarter**

We get a better overview by visualizing lead time by quarter.
Here we can see that product ideas released through Q1 2013 got out roughly 2x faster than product ideas released in Q3 2012. So, where did the improvement come from? Was it the simple case of just better understanding how to handle new technology?

**Where we made up the improvement in lead time?**

There are two components we can look at when we want to learn where the improvement came – waiting time and value adding time.

**Analyzing waiting time**

We can see how average waiting time is trending downwards. The higher up in each bar, the later in the flow the waiting time (and the more costly it becomes).
We lacked separation of data between waiting for customer usage and making customer adaptation. Therefore “waiting for customer usage” (in green) includes both waiting and sometimes value adding time.

**Analyzing value adding time**

Let’s turn our eye to value adding time.

![Lead time through development and system testing](image)

*Figure 14 Value adding time separated into development and system testing.*

Again we can see both time through development and test is dropping. The fact that development time is decreasing can be because we know the technology better, but we have also learned to prepare work and slice it better. But the biggest improvement has happened in testing - time through system testing has actually been reduced by a radical 7x.

**But are we shipping things of value?**

Flow matters little unless what we ship is of value. So how did we do in that perspective?

If you recall the last section of the kanban board you’ll note that it contained two sections: “popular at customer” and “oh crap!”. We collected statistics from these sections to learn if we had delivered things of value. Below you find the aggregated statistics:
Figure 15  Customer value feedback. Sample size is 113.

Happy means customer liked it and uses it. Dev rework means we stopped it before shipment, reworking the design before we deemed it satisfactory to ship to customer. Customer rework means customer where not happy, sending it back to us for redesign.
What’s different? Comparing now and before

While we do not work as a single cross functional team, a lot more work is happening in parallel today. Feedback cycles are faster. Before the first holistic feedback was returned at the late stages of system testing, today it comes already when the Concept is shared with the development teams.

Before

While we do not work as a single cross functional team, a lot more work is happening in parallel today. Feedback cycles are faster. Before the first holistic feedback was returned at the late stages of system testing, today it comes already when the Concept is shared with the development teams.

Communication content has changed. Before cross function communication (across function borders) revolved around completing specific activities, as defined per function. Cross function communication today revolves around what is needed to make the product work. This not only more frequent communication, but also different information is being exchanged.

Today

While we do not work as a single cross functional team, a lot more work is happening in parallel today. Feedback cycles are faster. Before the first holistic feedback was returned at the late stages of system testing, today it comes already when the Concept is shared with the development teams.

Communication content has changed. Before cross function communication (across function borders) revolved around completing specific activities, as defined per function. Cross function communication today revolves around what is needed to make the product work. This not only more frequent communication, but also different information is being exchanged.
What hasn’t changed?

So far team structures haven’t shifted - teams still by large are organized in functions along the stream (marketing, development, change management). Maybe this will shift in the future, but that is yet to see.
Summing it up

How far can you get by doing evolutionary improvement before altering team structures becomes necessary?

For us, we reached a 2x improvement in lead time over a period of 1.5 years. The main bulk of this reduction is due to less waiting time, better approaches to system testing, and better prepared inflow to development.

We have shown it is possible to let people passionate about ideas to run with them, regardless of role. (We don’t have the traditional roles of Product owners or Project managers). We continuously learn how this affects quality and usefulness of what we produce. For released products during the time period, 95% were reported value adding and useful.

Enterprise Kanban, a quality first mindset, focus on flow, ownership of result, teamwork and Concepts has helped us get to where we are today. It’s has been a fun and sometimes bumpy ride, we are no way near the end yet and I look forward to see how things move in the future.

Mattias Skarin, October 2013