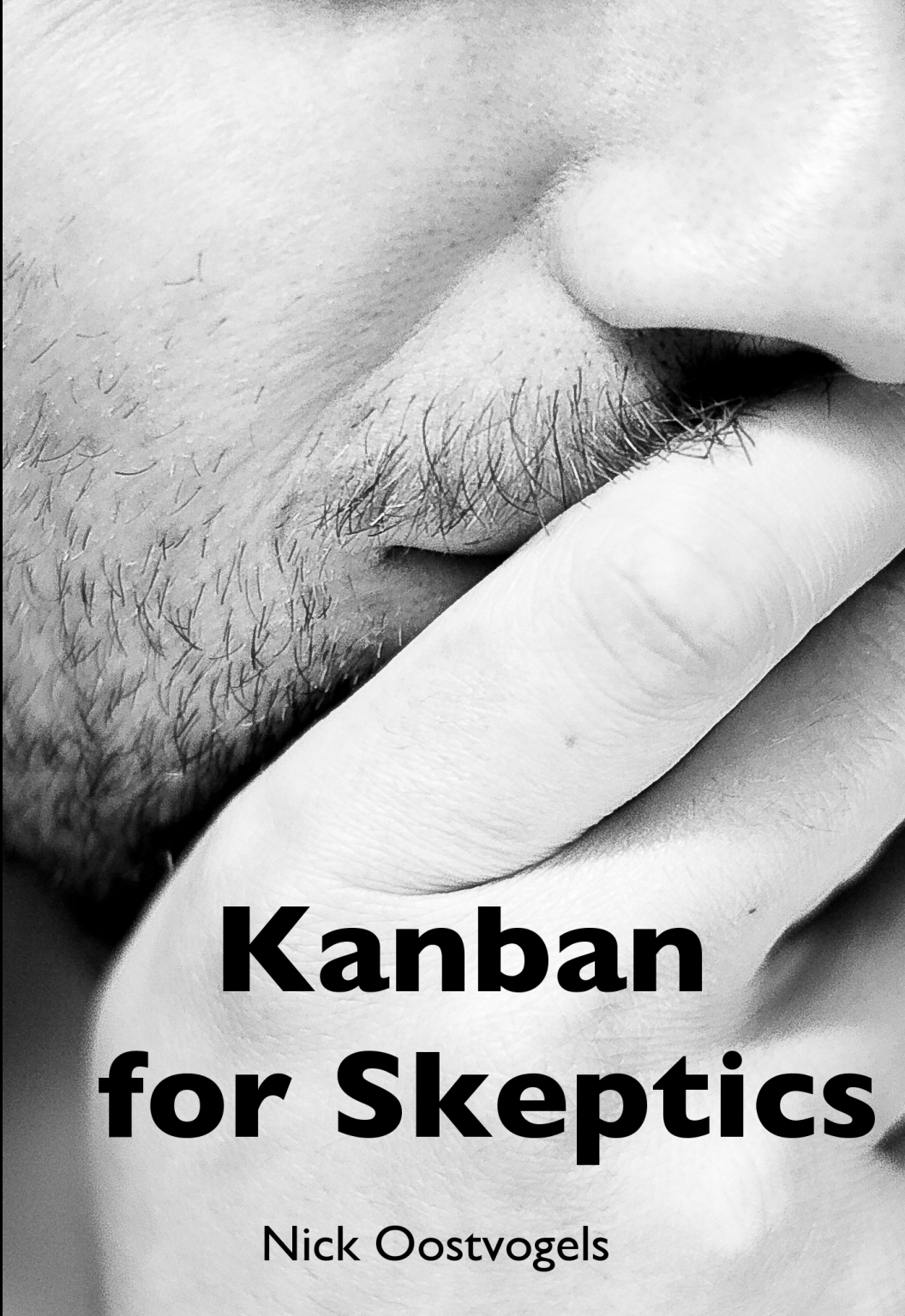


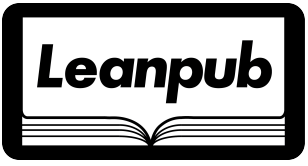
Kanban for Skeptics

Nick Oostvogels

Kanban for skeptics

Clear answers to Kanban in software development

This version was published on 2012-08-25



This is a Leanpub book, for sale at:

http://leanpub.com/kanbanforskeptics

Leanpub helps authors to self-publish in-progress ebooks. We call this idea Lean Publishing. To learn more about Lean Publishing, go to: http://leanpub.com/manifesto

To learn more about Leanpub, go to: http://leanpub.com

Tweet This Book!

Please help Nick Oostvogels by spreading the word about this book on Twitter!

The suggested hashtag for this book is #kanbanforskeptics.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#kanbanforskeptics

Contents

Acknowledgments

A special thanks goes to my wife, who makes it possible for me to spend all those long days learning and writing. That's not easy, when you have three energetic kids running around the house.

The one thing that annoys me about most acknowledgments chapters is the long list of names which the authors want to thank.

Luckily, I wanted to publish the first version of this book fast, so I only have a few names to mention.

A big thank you to Marcin Floryan and Jørn Hunskaar for reviewing the book so quickly in an amazing level of detail! I felt it increased the quality significantly. Thanks to all others who are still reviewing at this moment: Barry O'Reilly, Daði Ingólfsson, Jef Cumps, Jørn Hunskaar, Kris Philippaerts, Maarten Volders, Olaf Lewitz, Patrick Debois, Patrick Steyaert, Paul Klipp, Pawel Brodzinski and Yves Hanoulle. I'm sure with your reviews, the next version will be even better.

Two names keep popping up when I think about the people who influenced me to start writing: Scott Berkun and Jurgen Appelo. The stories on their blogs and in their books got me enthusiastic to start my own blog in 2009. It was scary at first, but has given me so much satisfaction in the past years, I can't even start to explain. Thanks for the inspiration!

The cover image[1] is taken by Jacob Bøtter and published under the Creative Commons license.

[1] http://www.flickr.com/photos/jakecaptive/3205277810/

Introduction

In my daily job as a change agent, I constantly need to reassure people that the path we follow is worthwhile traveling. This need is often expressed in the form of critique and difficult questions. When I coach Agile teams, this is often the case. The same thing happens when introducing Kanban. However, I noticed that Kanban raises much harder questions on a management and leadership level, once people are introduced to the basics and start to explore the subject on their own.

The type of questions Kanban raises, seem to be hard to answer without lapsing into an hour long discussion. I guess this is normal because Kanban is much less prescriptive than Scrum, for instance. In order to provide reassurance, as a coach, you need to trace the questions all the way back to the principles of Kanban, which are grounded in Lean thinking.

By listing the 5 most common arguments against Kanban and my response to them, I hope to help people in their Kanban journey and build great organizations that create amazing products. These answers are based on my own perspective and experience. It would be great to hear your answers and improve the book while more people are introduced to Kanban. The goal is not to explain Kanban scientifically, but provide insights why these arguments don't stand, in a language that is understandable by all.

1 What is Kanban?

If you mention the term kanban in an industrial setting, people will explain that it's a signboard which helps to facilitate a Just In Time production process. A kanban card is a token that is used to order a new part as soon as one is used in the process step. This creates a pull request that will ripple all the way up trough the process. The easiest example can be seen in a department store, where the purchase of a carton of milk can lead to a signal to order a new one. The major difference lies in the fact that customer demand is triggering the process (pull), instead of running it by forecasting (push).

When Kanban (with a "K") is used in the field of software engineering, it is used in a much broader sense. Kanban is a change management approach that employs a WIP limited pull system. For me it is an approach to change your process into one that focuses on end to end value and reliably delivering stuff, with high quality. The limited set of rules and principles help people focus on customer value and avoid building up half finished work, which is still a common issue in the software industry.

LimiteWipSociety.org has a nice description of Kanban: first follow the foundational principles

- Start with what you do now
- Agree to pursue incremental, evolutionary change
- Initially, respect current roles, responsibilities & job titles

then adopt the core practices

1. Visualize workflow
2. Limit Work In Progress
3. Manage Flow
4. Make Process Policies Explicit
5. Improve Collaboratively (using models/scientific method)

These few principles prove to be a very powerful foundation for an organization that drastically wants to improve its value stream.

Why is Kanban a change management approach and not a process? Simply because it doesn't prescribe how you manage your process in practice. All it does is change the rules that help your teams to focus more on end to end efficiency and quality. Because in the end, the only thing that defines success at an organizational level is your customer base. They don't care about your highly skilled roles, all they want is a quality product that is delivered in time. Roles or practices lead to change in the organization, no matter how broad their impact is. But because Kanban is not a process, people will perceive the change differently. In practice, the way they work won't change much in the beginning. The only thing they will feel in their day to day job are the principles that employ a WIP limited pull system. This will slowly introduce different behavior and continuous improvement.

So think of Kanban as much more than a task board with

cards and WIP limits. It facilitates organizational change in order to better satisfy your customers. This is very ambitious, but something most of us care about and can bring up the energy for!

2 We lose our ability to plan

The most popular criticism on Kanban is that it makes things impossible to plan. It is a natural argument to make, when you discover that features entering a Kanban flow are not estimated, but measured.

Think about the shock for a moment! For years we have managed our projects and people with the use of estimates. We have spent ages detailing out requirements and estimating them. There are people who specialized in estimating software features. Entire books have been written on how to do this better and faster. And now this Kanban thing is telling me it's no longer necessary! Yeah right, this must be another process created by lazy developers who refuse to estimate and take accountability. Why don't we just let them do whatever they like!

You see, planning serves two purposes. First of all, most customers need a date on which they can expect the product to be delivered. In some businesses, this date has to be more accurate than others. For instance, a car will be delivered near the end of april, but a tunnel repair must be finished at June the 5th.

Secondly, a plan is a tool for managing people. It might not be the best tool, but managers have used plans for ages to control their employees. "You said it would be finished by June the 5th, you'd better crank it up a notch if you want to make it." In combination with extrinsic motivators (like bonuses or fear of unemployment) a plan has been

the central project management tool for centuries. Let it come as no surprise that Kanban faces huge obstacles in mainstream management thinking. Let's take a look if Kanban actually does make it impossible for us to plan.

2.1 Release planning

A typical software development release plan is created at the start of project. The most senior people of the team take the time to carefully examine the specifications which have been sent to them through mail, by a potential customer. These folks have done this exercise a lot of times, but still it feels awkward every time.

They start categorizing the requirements and splitting them into smaller chunks. At the same time, a technical architecture is created which matches the demands of the system. The analysts start translating the customer requirements into a format that is understood by developers. A couple of weeks later the entire specification set is being transformed into formal requirements and sent to a senior developer who estimates the entire collection.

Off course summing up the estimates is quite discomforting, when they realize their potential customer will never agree on such a big price. So they gather around the table once more and try to find ways to cut down the estimates. The senior developer argues that the estimates were already optimistic, but he folds under pressure of his superiors and reduces the estimates. After some final commercial negotiations, both parties agree on the contract.

At this point we have a release plan, based on estimates, which will become the focus of a lot of people, in the not too distant future. This seems like a normal approach. The customer wants something from you, you ask him about the details, he asks you how much it costs and when he will get it. Supply and demand in its most basic form, right? This works quite well when you can rely on a repeatable process, like building a car. At the point when you order a new one, the car company has already produced thousands of them. They have tuned their assembly lines and can give you a pretty decent estimate of the delivery date. This is because they know how many cars are being produced at the moment, how long it will take until your car gets on the assembly line and because they know how long it takes to produce one car. It is repeated over and over again until that line is retired.

When we try to do this with a custom piece of software, things are quite different. At the point when you order a new system, the development company, in most cases, has never built a similar system. But still they treat it like a car manufacturer would. However, instead of estimating the work based on measurements, it's based on guesses. You might argue that experienced developers are good at estimating, because they have many years of experience in the trade. Maybe, but it remains a guess because they do not have any knowledge about the future. Their experience can only make it a more educated guess.

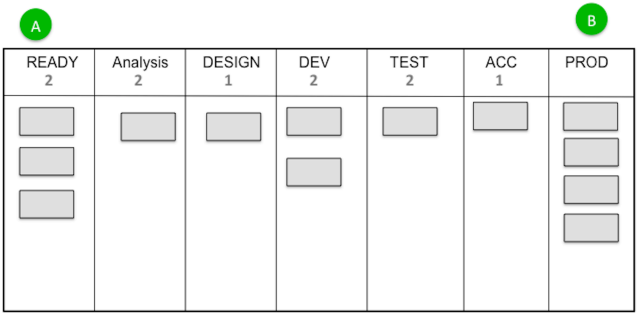
To make it even more depressing, software development is a human trade, a creative process, which makes it even harder to predict. We've all been part of a team where

small variations in behavior cause unpredicted results. For instance, when a conflict between two team members cause them to stop speaking to each other resulting in less cooperation and a delay in schedule. Your plan is that fragile! A sneeze could theoretically (and physically) mess up your entire project.

Measuring

Let's forget about this release plan for a moment and try to look at things bottom up. A car manufacturer can tell me with good accuracy when my new car will be ready. Remember that they can give this information only because they measure how long it takes to produce a car. Thanks to the repeatable nature of car assembly, these measurements are an effective tool to provide fairly good estimates.

The Kanban method starts from that angle as well. Instead of trying to build a release plan top down based on estimates, Kanban teams base their plans and actions on measurements. Like measuring the construction time of a car, the time it takes a feature to move trough the flow is measured in Kanban.



Measuring flow from point A to B

If we can measure how long it takes for a feature to move from point A to point B, we can use that information to provide good estimates to our customers.

The calculation is simple. How many features are waiting in front of it in the queue? Multiply that by the time a feature takes to get from point A to point B and you've got yourself an estimate. Here we are calculating the lead time of my new feature. But if our Kanban flow has a throughput of 2 features, then our lead time will only be half that long, since our queue will empty twice as fast.

Sounds too good to be true, right?

I hear you thinking : "How can I work on an order from my customer and give him an offer by using measurements, when the project has not even started?" Well that's easy, you can't! You can't without estimating your features. And we're back to square one.... Not completely! There are 2 easy ways to do this without falling into the trap of

estimating and using the power of measuring.

1. Use a scale
2. Keep your features small

Using a scale

Humans are quite bad in coming up with precise estimations, but we are pretty good at comparing one thing to another. I always use the following example: When I have to drive to a new place for the first time, I have to guess how long it would take me. Coming up with this number from the top of my head is impossible, even if I know the distance. So I start looking for references. Is there anything similar I've already driven too? Let's say I had to drive to Mechelen, which is further than Antwerp, but closer than Brussels. Lucky for me I've driven thousands of times to Antwerp and Brussels, so I have some points of reference to start my comparison. Mechelen happens to be half way between Antwerp and Brussels.

Estimating a drive from home to Mechelen

At that point I've placed Mechelen on my experience (or measurements) based scale. Today an Antwerp point on my scale equals 45 minutes. It used to be less when traffic was still ok, but that's the beauty of measuring, it's always reflecting reality. Without having ever driven to Mechelen, I can now come up with a relative estimate, just by putting it on my scale. A scale that makes sure my estimate is based on reality, and will therefore be quite accurate. Now what

kind of scale could we use for software features? It doesn't really matter. Whether you use t-shirt sizes, story points, letters or fruit types, the point is that a scale helps you to size features without having to care about the amount it takes to deliver them. This keeps you away from the release planning issues we've seen before. Everybody can compare features, right? Some are smaller or easier than others, you might have big features, or medium ones.

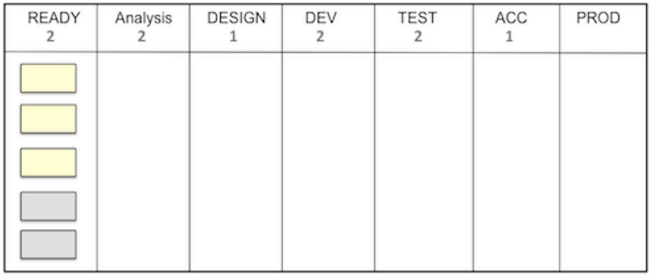
Once you've placed your features on a scale, you can calculate the time it will take to deliver them, if you have measurements. You can pretty easily come up with an average time it will take for each point on your scale to move through the flow. Just be aware that you're still maneuvering in the dark, until you have your first set of measurements. And of course building software features is much more variable by nature than building cars. We'll see later how to address this.

In an environment where you have mutual trust between you and the customer, it's in both parties best interest to wait for the first measurements. We both know that in software development, reality is always different than what we expected. If you're in fixed price setting, you might want to start with a first short term contract in which the goal is to get a set of measurements. Both parties can then decide whether the project is still interesting. If the outcome looks bad, at least you know fast enough to stop wasting any more budget.

Keeping your features small

By splitting your features until they all fit a certain size, you can benefit from statistics.

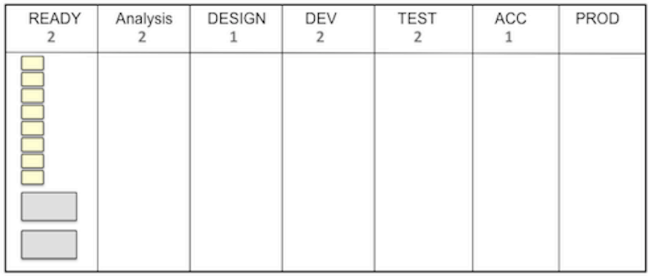
Let's say we pick one feature as a reference point and we try to size all other features in a way that they are never bigger than our reference. This is even easier than putting them on a scale. The result is you end up with a good number of features. And this is where you will benefit from statistics, more specifically the law of large numbers. Let's look at an example. Suppose the yellow post-its are features we have distilled from our customer's requirements.



Large features waiting

We feel there is a lot of difference in size & complexity of these features. Our next step is to split them up until they feel like they could fit into 'a couple days of work' (don't just think of this as a day of programming, also include other steps you need to perform to deliver the feature). Use your gut feeling during this exercise. There's no point in

discussing how the features will be implemented, right now you're just sizing the list of user needs. This is what we end up with, almost 3 times the number of features.



Splitting features

Planning with measurements

You can now create a release plan based on the length of your flow, the number of features and the throughput. How the hell can this be accurate, you ask?
Well, lets look at what the law of large numbers tells us.

In probability theory, the law of large numbers is a theorem that describes the result of performing the same experiment a large number of times. According to the law, the average of the results obtained from a large number of trials should be close to the expected value, and will tend to become closer as more trials are performed. When you do the exercise by rolling a dice, the more time you roll it, the closer the average will get to the average of the points on the dice (which is 3.5). If you keep your features under

a certain average, the more features that pass trough the flow, the closer you will come to that average.

Would we see this behavior with our initial set of features? What's the point of splitting them into smaller pieces? There are two reasons.

The exercise of splitting them up forces you to get better informed about the content of the feature.
You need to get a little deeper into the needs of the user to be able to find a way to split it. This increases your knowledge and reduces errors in judgement without going all the way into an up-front detailed analysis.

You end up with more features which is important for the law of large numbers to have an effect.
The more points to measure, the better the law of large numbers will average your results. By keeping the upper limit low, the variation in lead times of different features won't annoy anyone and you can just use the average to plan.

When you think about this, Kanban is even more appropriate to manage large projects. Remember those projects that make the news because budgets trippled and get delayed by years? The size of their scope is so large that they often work with large teams and vast timings. Remember that a sneeze can theoretically mess up your project, then risk gets exponentially higher because of the larger number of people involved and the multi-year planning. Instead you could try another approach. Leave the master-plan behind, split a first set of your features and measure the flow. Chances are much better you will soon reach a point where a first version can be released, and

you're able to tell your stakeholders when!

Reduce variation

A critical success factor for Kanban which many tend to forget is reducing variation. Implementing a pull system will lead to a continuous flow. And this means that it needs to provide value in a regular cadence, for these reasons:

1. Working with averages is only comfortable with limited variation. If you're keeping features small, you have to reduce the number of times a feature takes much longer than the average to travel through the system. Otherwise, even if it's only psychological, you will be pulled back to the estimation game, solely because a perception grows that the system in not reliable and estimations are needed to assess the difference.

2. The benefit of a continuous flow for many stakeholders is its fast response. You're able to re-prioritize the queue at any time. Whenever an urgent request gets in, you can consider moving it to the top position of the queue. The natural reaction would be to get hold of a team member, make him stop whatever he's working on and give him the new assignment. But we know that this carries a great cost. Context switching, parking half finished work, lack of personal motivation,... all lead to waste. In Kanban we respect rules that prevent this behavior, but only when we can rely on its promises. When I come with

an urgent request, and I know the team takes a new feature from the queue every 4 hours, I can make a conscious decision about prioritization. If I cannot rely on this, because of variation, the system will not hold and you will quickly fall back into old habits.

3. Big variation in lead and cycle times indicates a flow that is not under control. It's easy to feed a flow and measure the outcomes. Any team is able to switch to this model without big effort. But then the only thing you've done is visualizing your process and defined some extra rules, like work in progress limits. The point of introducing WIP limits is to start the evolution to a continuous flow that is consistent and delivers value on a regular basis at a speed that matches the needs of your organization. To be able to do this, your flow needs to be continuous. It needs to be reliable with limited variation (I'm not saying NO variation) because that's one of the indicators that you've improved your process and have it under control.

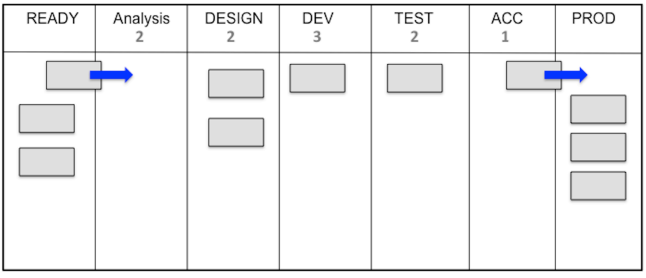
4. Continuous improvement is only effective in a stable process. The strength of Kanban lies in it's focus on evolution. There is no manual you can use to get started. There are only a few ground rules that help you evolve your own Kanban system, in your own organization, with it's own difficulties and culture.

These ground rules stimulate everybody involved in the process to think about improvement. In order to be able

to evaluate process improvements, you need to compare measurements. Ex. "What effect did the lower WIP limits have on our cycle time?". It is hard to draw any conclusions if your flow behaves like a crazy rabbit. Any change you see in the new measurements can be a coincidence. You might even see the opposite to what you were expecting which make you draw the wrong conclusions. That's why a first step in continuous improvement should be to reduce variation. As soon as your flow is delivering consistently, you can start to initiate improvements and work towards a Kanban system that matches the expectations of the organization and customers.

Small releases

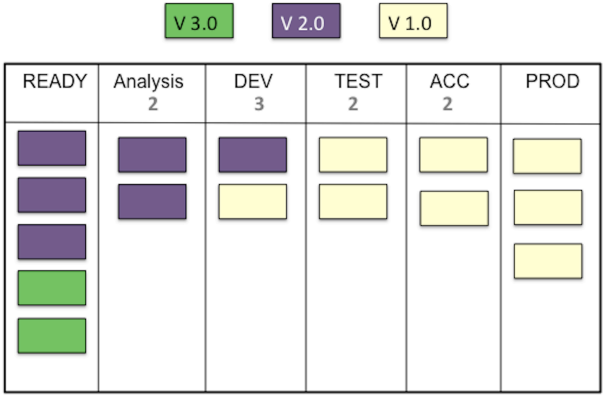
Many people think Kanban goes hand in hand with continuous deployment. It doesn't! Continuous deployment will lead to a Kanban-like process, due to the nature of it's rhythm. If you're deploying one feature at a time, working with large batches makes no sense. It might be that the middle part of your flow contains batch-like stages due to bottlenecks, but the input and output will come close to a one piece flow.



One piece flow at input and output

A Kanban flow doesn't necessarily need continuous deployment. In many organizations, this is a model they are not able to achieve yet. Others don't see the benefit. They need a full set of features before they can release something to their users. It's not an easy task, finding the right set of features that need to be in your first release. Product managers often get tempted to include a few too many. This is related to the uncertainty of a product launch. "Will people like our new product? Maybe we should include these 5 features too, just to be sure." Fact is that increasing the size of your first version carries a great risk. The risk of spending too much resources on the wrong thing. You can test your product with 1000 people in focus groups, distribute surveys, etc, but still fail to gain market share when you go live. Why not keep your first version small, spend a modest amount of resources and find out quickly if your product has a future? Eric Ries explains this concept in his book 'The Lean Startup'.

This is where Kanban is a great match. Pull your features one at a time and release your first minimal set. If the reaction of your users is promising, continue to add features on a regular basis, by pulling them into the Kanban flow. This way your return on investment starts way earlier, which will catch the interest of your CFO.



Minimal versions moving through the flow

"But won't this annoy our customers? Having to update every couple of days?" I think we can all relate to this. Getting these bloody updates imposed, which mess up your entire system…

In recent years, another type of update approach has appeared. Hip new online applications releasing new versions each day, often without the knowledge of their users. I believe this is an approach that has great potential,

for these reasons:

1. It doesn't annoy your users. These update are so small, that you hardly even notice them. Each day a small part of the application changes, which you probably notice, but never think about. You are actually changing with the application and start using new features without even realizing it.

2. The risk of bugs is lower. By keeping releases small, they can be easily tested. You can do an impact analysis and focus on the impacted areas of the application and test them thoroughly while the rest of the application is tested by a set of standard regression tests.

3. Releasing early creates a sense of urgency. As long as the product is in development, you're living in a safe little cloud. Exposure to the outside world is limited in the form of occasional demo's and maybe some marketing material. Your biggest concerns are company related, in the form of internal politics, team dysfunctions, budget wars,... As long as your product hasn't been launched, these are serious threats to a successful release. A lot of projects get cancelled after having spent a considerable amount of time and budget. This threat disappears almost entirely when the first release is on the market. Suddenly other concerns start to surface instead of internal company issues. Things like bug-fixing, support, customization, professional services and sales swipe

the old concerns right off the table. To summarize it, you can finally start the real work!

2.2 Re-planning

We've seen how you can plan a release in Kanban, but planning doesn't stop there. Once things are moving, we need to continuously re-plan, whether we're doing Agile, Waterfall or Kanban. Agile has its iteration planning, Waterfall has its status meetings and change control procedures, how does this work in Kanban?

Well actually, it's entirely your choice. You can decide which approach works best in your organization. Here are a few examples.

Re-prioritizing the input queue

It's the input queue that's feeding the process. Priorities set in the input queue determine how quickly a feature will be delivered to the customer. So suppose you have a new feature that needs to be delivered as fast as possible, what do you do?

It's important to look at the input queue and not blindly insert the new feature at the top. Prioritization in Kanban looks easy, but can have serious consequences when done without serious thought. If urgent requests always get inserted at the top of the queue, you are endangering your product vision and constantly changing course. You'll end up with happy customers in the short term but limited future for your product in the long term.

That's why it's important to create prioritization rules on which you can rely when pressure is high. There are several questions your prioritization rules must be able to answer:

1. How much % of our capacity do we want to invest in development of the product?
2. How much % do we want to invest in specific customer requests?
3. How much % do we want to invest in technical requests?
4. What priority get the different categories of bugs?

I'm sure you have some other, specific to your situation. Think about the tension between short and long term. By comparing the current reality and the road ahead, you can make a conscious decision on how you want to use your people and find the right balance.
Without a clear set of rules on which all stakeholders can agree, prioritization will lead to many discussions in which the loudest voice will often determine the result.

Cadence

Kanban is based on continuous flow, but this doesn't mean you can't have iterations. If we think about an iteration as a heartbeat, a synchronization moment, then this can be applied to a continuous flow as well.

Many organizations find it useful to have a fixed moment every x weeks to demo the new features that have been developed. Most of the times, this is related to practical issues, when not all stakeholders are flexible enough to evaluate each individual feature that is waiting for feedback.

Together with the demo cadence, they take this opportunity to evaluate the remaining items in the queue, since all stakeholders are present anyway.

Because prioritization is really important, and features have different requesters, they must reach an agreement on priority together. This is also the moment when feedback from the demo gets its way into the queue.

Be aware that this will introduce a larger batch size into your Kanban flow, since features are queueing in one of the last workflow stages waiting for a demo and acceptance. The same remark applies to the prioritization of the input queue which no longer happens each day, but every x weeks.

Pulling a planning meeting

Some teams prefer to plan a number of features together instead of doing them one by one. Reasons are often related to knowledge sharing, where the input of the entire team has higher value than they would get if planning a feature with a limited number of people. Sometimes it's a first step to move from an iterative process to a continuous flow.

The process goes as follows. A queue has been introduced right after the input queue for development. This

ready queue has a limit, depending on the size of the development team. Whenever the ready queue gets empty, they pull a planning meeting with the entire team. This is where they take the next items of the input queue and talk them over with the team. You can compare this with an average sprint planning meeting, where a product owner explains the next set of features. The team discusses and asks questions until they are comfortable to start breaking them up into tasks. Complex features may require some design decisions, others don't. At the end of the meeting the team has filled their ready queue, which becomes their focus for a next period in time, until it empties, after which they pull another planning meeting.

2.3 Summary

As you notice by the length of this chapter, there is no easy answer to the argument 'We lose our ability to plan". Remember the main points, classified in 2 parts, release planning and re-planning. This gives you two big parts in your answer.

Release planning is still possible by measuring instead of estimating. You get the ability to pursue small releases that can have major advantages by reducing risk and getting faster feedback. By reducing variation and continuous improvement, release planning will become easier and more reliable. Instead of drawing a perfect world, we use reality to make better decisions. Re-planning is wide open in Kanban. You can choose to work with a cadence or be

even more adaptable by constantly reevaluating the order of the input queue. But remember, with great flexibility comes great responsibility. Re-arranging needs to be done consciously, in agreement with all stakeholders, by using a clear set of rules which are distilled by the needs of the organization and its customers.

3 It will take longer

A second popular argument against Kanban is that it will take longer to create the product. This reasoning is deeply entrenched in our everyday life. It is related to our education and how business have been managed for centuries. We've seen that Kanban focuses on using real data to make decisions. By measuring flow we manage our people and product.

This is completely upside down to what we're used to. Managers have learned to control their workers by assigning work and monitoring their performance. You get a task and get it finished on time or you have to justify why you didn't make it on time. The same approach is taught in schools. Students get an assignment with a deadline, which needs to be finished on time or they'll feel the consequences through punishment in the form of lower grades. How many times did you get an assignment without a deadline? "This is what I'd like you to do, don't worry about the timing, we'll measure afterwards." Let's find out how Kanban uses measurements as a core management tool without losing focus.

3.1 Parkinson's law

"The amount of time which one has to perform a task is the amount of time it will take to complete the task." Many people act by this simple definition. If we don't set any deadlines, a worker will take much more time to complete the task. He will relax and start working

whenever he feels like it. Time is money, so we can't allow this, right? It has been done like this for decades in the world of project management. A project plan is created by the project manager and senior developer. Based on estimates, a schedule and timing is created which becomes the target for a development team, whether realistic or not. The reasoning is quite simple, the team needs a deadline to keep focused. The project manager will add some buffer in his own plan, just in case the team doesn't make it. But then at least they will make it for 90 percent. Without this focus they might not even reach 50 percent…

Simply from a cost perspective, this is a good approach. From a value perspective, this is a bad approach. Because what happens when people get stressed? They cut corners, try to do more in less time and introduce malfunctions by sloppy work. From an HR perspective this is also a bad approach. People get overworked, stressed out and leave the company if the external pressure is too high. You realize that there needs to be a balance between freedom and control. A first step in project management was to include the team in estimating the project plan. This gave them a first voice in the management side of the project and resulted in an increase in intrinsic motivation. Later with agile development, a next step was taken by measuring what a team could deliver in an iteration and using those numbers to plan instead of pushing for an initial estimate. This led to a significant increase in value creation and quality. How does Kanban deal with keeping a healthy focus between control and freedom?

3.2 A healthy balance in Kanban

In a Kanban flow, team members pull new features from an ordered input queue. In many cases these features are not estimated up front. At most they are sized on a scale. Measuring flow is the main management tool that is used to control efficiency. By continuously monitoring lead and cycle time, we get a good idea about the progress of the team. These numbers become a target for each contributor to the flow. If the average cycle time is 5 days at a certain point in time, the team will indirectly focus on this number when starting on a new feature.

The beauty in this approach is that it's founded on real data, not wishful thinking. If it takes 5 days to deliver a new feature, chances are high that it will take approximately 5 days for another feature. The focus is not created externally by management, but it has evolved from within the team. Therefore they will be more intrinsically motivated to pursue it. People will have to cut less corners and will deliver more valuable and high quality features.

From a management point of view, the style changes from command and control to helping the team to improve. Accepting that the system can deliver a new feature in 5 days is accepting reality. From that point all involved in the Kanban flow can start the process of continuous improvement. Everybody is responsible to think about how we can improve flow, reduce bottlenecks and variation. Soon your average lead & cycle times will change, along with the focus of the team. Their new focus will be a realistic one since it is based on real measurements on an

improved flow.

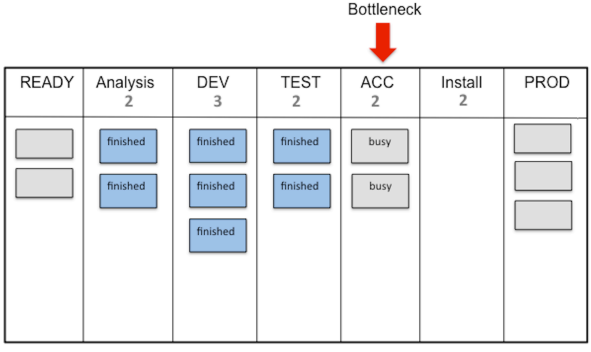
I found it very useful to keep the theory of constraints in mind when trying to improve your Kanban flow.

3.3 Theory of constraints for process improvement

The theory of constraints (TOC) is a management philosophy introduced by Eliyahu Goldratt in his 1984 book[1] titled "The Goal". It is based on the idea that "A chain is no stronger than its weakest link". If we project this to Kanban, this means that the weakest stage in the flow will determine the rate of value creation of the entire system.

For instance, if we notice that the user acceptance testing stage can't keep up with the preceding stages, it will slow down output of the entire Kanban flow to it's own capability.

[1] http://www.amazon.com/The-Goal-Process-Ongoing-Improvement/dp/0884270610



The Acceptance stage as a constraint

By visualizing work on a Kanban board, the TOC will become visible in the form of bottlenecks that block the system. During the course of their work the team will also feel the TOC trough the use of WIP limits. For instance if we've decide to start with a WIP limit of 3 on the UAT column, this means that no more features can be started in the preceding stages if the UAT column if full. You can imagine that this can become quite frustrating. Most of us are eager to start a new task, especially when we're trying to get a release out of the door! People feel uncomfortable doing nothing, because we think it gives a bad impression. We must be busy all the time! It won't take long before a situation like this is escalated. But that's a good thing, we want to find bottlenecks as soon as possible.

This is usually the moment when an improvement ses-

sion is triggered. How can we prevent this from happening again? Are our WIP limits not set accordingly? What is the root cause of this bottleneck, how can we unclog it? This session resembles a retrospective, but with a wider focus. Most retrospectives are held within the development team, with an occasional participation from a product owner. In Kanban, improvement sessions typically include many more different roles of the organization because of the focus on the entire flow, from idea till installation on production. This causes business analysts, system engineers and sometimes even sales people to be included.

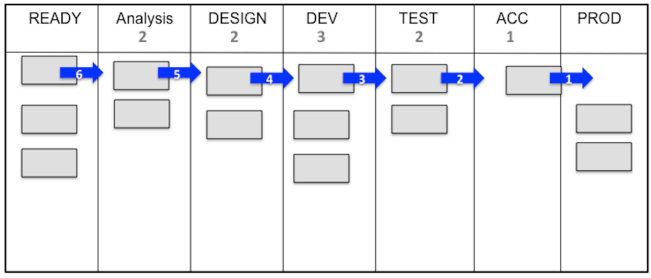
The continuous improvement focus of all the different roles from the organization creates a healthy peer pressure and more opportunities to share knowledge. WIP limits will reveal bottlenecks quickly and create momentum to help others and start the continuous improvement cycle. The power of continuous improvement in Kanban will help you improve flow, which will tilt the perception that it will take longer into a feeling of greater efficiency. And you will have the measurements to prove it!

3.4 Flow

Kanban is based on the principles of a pull system, pioneered in Lean manufacturing. The goal is to get as close a possible to a single piece flow, where the organization is only working on products that are ordered by a customer. Why is this, and what does this have to do with software development?

The reason we only want to be working on customer requests is because we want to eliminate waste. Forget about innovation for a moment, because there's no customer order is triggering the pull system. Working not only on customer requests means you're guessing what the customer will order in the future, and therefore you are building up inventory, a stock of products waiting to be purchased. This is waste because of two reasons. It is capital spent on items that are not bringing in revenue and therefore costing us money. Also, these items contain a big risk of not being sold or not complying to customer's wishes, leading to costing us money. In software development the same issues apply. Features waiting months to be released are costing money and contain a risk. You might argue that most work in software development isn't ordered by a customer, for instance when you're still working a first version. Even then, those features have a customer, being a product owner, a business analyst, senior management, a potential customer segment, …

WIP limits reduce inventory and create a pull system upstream. If you have finished a task, you pull a new one from the preceding stage, freeing up capacity there, giving an opportunity for the preceding stage to pull on its own. This keeps going all the way up to the input queue.



Features getting pulled through the Kanban flow

The most common critique on WIP limits is that it will cause a drop in efficiency. Off course, people will be idle once in a while because of WIP limits that are reached. What would happen if they would ignore the WIP limits? The organization would gradually start building up inventory, more than the flow can handle. People can't be 100% busy. The perception that they should can be traced all the way back to the start of industrial times when factories started to grow and expensive machines were introduced to increase capacity. Off course these machines were so expensive that they had to run continuously in order to keep the unit cost low. This led to large unevenness in the production line and a growth of inventories. In those times, a product was not customized to the user, so the risk of not complying to it's wishes was none existing simply because they had no choice. The company just had to make sure they sold the inventory. Today, for many organizations products are customized by the end users,

causing inventory to be a major risk. This applies for IT as well, since the majority of systems are customized for a client, a customer segment, …

3.5 One continuous sprint

Kanban being a continuous flow leads to the perception that teams get no more chance to pause and reflect. It is a never ending sprint where team members are pushed to deliver faster and faster. As we all know, a sustainable pace is important for the quality of the end product and the health of the team. Some slack is needed to get your senses together or reflect on the future of the product which allows the team to take better decisions during the course of delivery.

Measuring flow makes it easy to fall into the trap of micromanagement. Pushing the team to work harder, feature by feature. This is where a coach has an important role to play. He must help all team members to understand that in order to go faster, they must take the time to reflect on the process, not on the speed of development. Kanban focuses on the end to end flow. Pushing one step in the flow to go faster, will often cause problems for the rest of the flow in the form of defects, bottlenecks and swift decisions. This is again the Theory of Constraints which tells us that problems will appear downstream if we push one step in the process to go faster.

A new Kanban team starts with their current efficiency and continuously inspects its collaboration in order to

perform. Not by working faster, but by optimizing the flow. This makes it easier to guarantee a continuous flow, since we all agree that the system is more important than the individual, and we can achieve more by improving together.

3.6 Summary

It is a strange idea that by measuring flow instead of setting deadlines, work will be done efficiently. Kanban creates awareness on the importance of every phase in the process. Instead of pushing each individual phase to go faster, we optimize the whole. Again, because this is what matters to the end users. They don't care how well our developers are meeting their deadlines, they need the order to be delivered. As soon as the process is under control, the team can start trying to improve their efficiency. Not locally, but end to end. Not by pushing individuals to work harder, but by reducing bottlenecks and optimizing the flow. The circumstances in which people work are tuned to better suit everyone's needs. A sustainable pace for the employees, a reliable relationship for the client and a profitable business for the owners.

4 Things will get stuck, we can't keep WIP limits

When people hear about the concept of Work In Progress limits, their first reaction is that it will not work in their organization. They expect a major efficiency drop if they would apply it to the different work flow stages. "Our testers can never keep up with the pace of our developers when testing the features. Developers would be idle for half of the time". This is applicable to all different roles in the organization all the way up from sales to operations downstream. The way most organizations fix this, is by working asynchronously. When testers are looking at the features, development is starting to work on new requests, to maximize their efficiency.

Remember that Kanban doesn't focus on maximizing resource utilization. Instead it focuses on maximizing end to end flow efficiency, which is a completely different goal.

4.1 End to end flow efficiency

Efficiency has been a management obsession for centuries. This is based on basic economics such as supply and demand. I look for extra manpower, you offer your services at a monthly price in the form of a wage. As soon as services or products are purchased at a price, the natural reflex of negotiation starts. Both parties want to get the

most for their money. It is normal that an employer wants as much produced for the wage he pays. So keeping the employee busy is the aim, because otherwise he would be paid to do nothing.

The employee wants to get the best benefits he can get for the job. So he will negotiate new terms on a regular basis, driving his cost for the employer up. From a cost perspective, the employer is motivated to drive up efficiency, making sure the worker spends as little time idle as possible. This has serious consequences on end to end efficiency. Imagine a shoe factory worker producing as much shoelaces as possible. Each day he works for 8 straight hours, producing a huge pile of shoelaces. Chances are high that he produces a different amount of shoelaces than required for customer orders. No problem, these can be stored in a warehouse and used later, when sales go up. Remember that this increases inventory, and therefore cost, in the sense of no return and extra storage capacity investments. These issues can be dealt with, for sure. Think about this scenario. What if he produces different colors of shoelaces? His increased efficiency causes the company to predict colors of shoelaces because of the 100 he produces per day, only 80 are linked to a customer order. In the best case these 20 extra shoelaces get linked to a customer order later. If not, you 're ending up with a stock that is causing money and cannot be sold at regular prices.

Driving efficiency from a cost perspective will make it impossible to run your business as a pull system, that works on customer request and reduces waste. It will shift an organization into forecasting and predicting sales. As

we all know, predictions seldom come true, no matter how advanced your forecasting techniques are. This is even more troublesome in software development. Analysts who are forced to maximize their time on analysis from a cost perspective, will start making assumptions about useful features for customers. They build up an inventory of features that may or may not cover their customers interests. Unlike in manufacturing, features are intangible and subject to changes in the market. An interesting feature today can become obsolete overnight, creating waste of inventory and inefficiency.

The focus of Kanban on end to end efficiency all points towards pulling value from the customer with as much added value as possible. So instead of maximizing utilization rates, the team focuses on getting value through the flow as quickly as possible. In some cases this may cause people to be idle, making sure they are not working on things that cause an increase in work in progress and the risk of a value mismatch. As you can imagine, the pull system is also subjected to change. A change in sales can demand extra capacity from the system. But instead of all blindly starting to produce as much as possible, the team focuses on improving the flow by removing bottlenecks and redesigning the flow in order to keep up with the new stream of demand. This is an exercise that never stops, due to the variability in the consumer market. Thanks to the clear focus on end to end flow, the team makes sure they keep working in a pull system, driven by customer demand.

If we're honest, we all know that it's in our nature to start looking for new challenges when the major problem

solving of a task is done. This leads to a huge pile of tasks that are only 90% finished. The end to end focus of Kanban will push you to finish the remaining 10%.

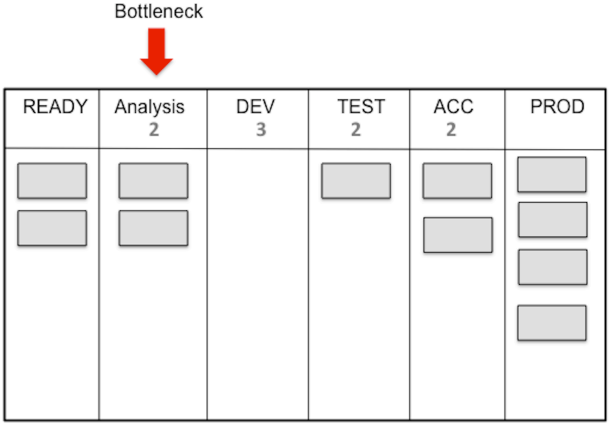
4.2 WIP limits will always cause bottlenecks

There is no exact science on determining WIP limits. As soon as they are chosen, bottlenecks will appear. This doesn't have to be a disaster. When the organization embraces the idea of a pull system, they will understand that WIP limits help to improve end to end efficiency. The first signs of a bottleneck appearing will drive continuous improvement. People will start to get nervous because they can't pull a new feature to work on. This is a very uncomfortable feeling, not being able to work due to another stage that can't keep up. It will drive team members to collaborate on 2 levels. A first reaction will be to go out and check what the problem is. Why are you stuck? Can I help you to get going again? This will eventually get the flow going again, but soon things will get stuck again, leading to the second reaction. How can we solve this bottleneck? Is it just a matter of uneven capacity between stages? Do we need to correct the WIP Limits? Can we add extra people or do we need to look for other options? The team starts the improvement cycle and together they try to agree on actions. If you think about it, the WIP limits cause the team to self adjust to changes that affect the end to end flow.

Let's look at the most common botllenecks that surface in software development thanks to the new WIP limits.

Requirements aren't ready

A Kanban flow generally starts with the steps needed to transform an idea into features that are ready to be created, just like any other traditional process. If we introduce WIP limits on these stages, often the upstream stages can't keep up with the down stream stages. Developers are finishing features, and eager to pull new features.



Requirements aren't ready to be pulled

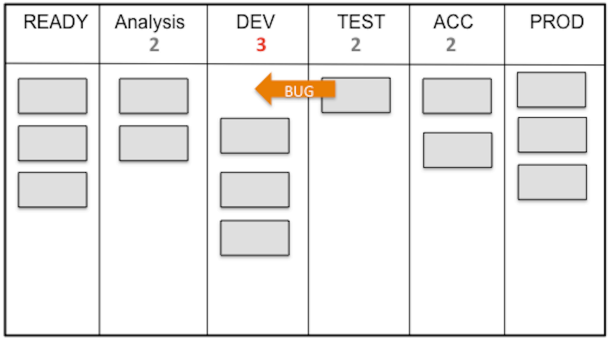
Why is this happening? Is it just a matter of a mismatch in capacity? That's possible, but it is something

for the team to find out in their improvement sessions. The analysts may just be adding too much detail to a feature. This is something the team can evaluate during the improvement session. Which actions are taken in the bottleneck stage? Is there any waste we can reduce? For instance by providing less detailed information and work out the details via communication when a feature is pulled into development.

Another cause is often an outside dependency that is not visible in the flow. The analysts may need to go trough an approval board that only meets once a week, causing them not to be able to guarantee a smooth flow downstream. It is by tracing the different steps in a bottleneck flow that the team discovers room for improvement.

Bugs cause features to be stuck

Quality control is an important aspect in product development. Customers demand a working product without any malfunctions. Teams that switch to Kanban often see a big risk in quality control as a blocker for continuous flow. They know from experience that the test team will always find an issue, causing a feature to move back into development for a fix. This will cause the WIP limit to be broken, because of course the developer has already pulled a new feature to work on.



bugs cause rework

This doesn't have to be a drama. As soon as this happens, the team should have a natural reflex to drop what they started on and fix the feature immediately. Removing the WIP limit violation as quickly as possible and get the feature flowing again. Flow can go both ways in Kanban, there's no rule against it, but this often indicates there's an opportunity to improve. Why do features flow in circles? If the team can work out an way to reduce the amount of returning features, flow will improve dramatically.

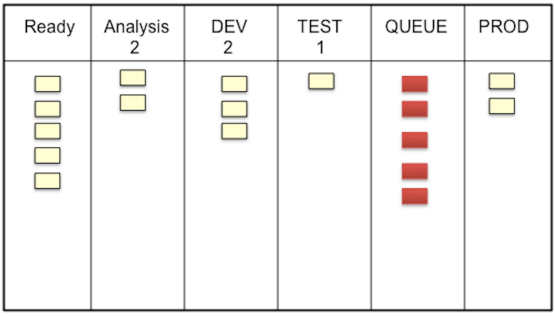
The WIP limits turn quality issues into an uncomfortable experience. In the old days, bugs were just listed and prioritized, but often people just carried on with the next task, fixing bugs only every couple of weeks, or (even worse) at the end of the project. In Kanban bugs cause the flow to be interrupted and people to get stuck because of WIP limits that are reached. It's no longer possible to sweep

this under the carpet. If quality is not reaching our defined standards, it must be fixed immediately.

Think of the change for a minute. Where in the past, quality standards lingered in the back of our head, they now kick us in the face, for each feature if we're too stubborn to learn. Again Kanban proves to be a great change management enabler. No coach is telling us to focus more on quality, the system is showing us the issues and lets us experience the consequences. This is much more powerful, as you can imagine. Because the team drives its own improvement, they have the space to do something about it. That's why its so important to be strict about WIP limits and quality standards. They will help the team to see and improve.

Long deployment cycle

Some teams have the Kanban flow well under control. They are able to pull requests on a reliable cadence, keeping work in progress limited and being reactive to change. However, what really counts to the customer is the end to end flow. From the time of the request, until the time it is delivered. In many new Kanban teams, the last stages of the flow, being installation into the production environment, soon become a bottleneck.



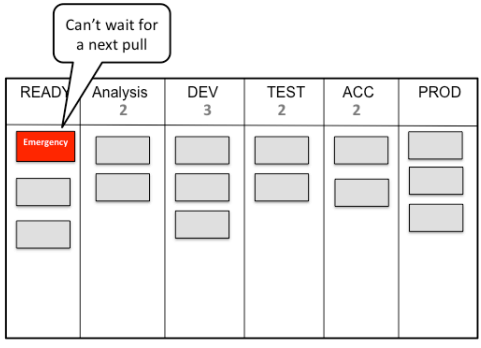
Features waiting to be deployed

Removing this bottleneck will require bringing operations and development closer together. How can we release more often with high confidence? Are we technically able to release fast? Can we evolve towards a model of continuous deployment? Otherwise increasing the number of releases will bring extra overhead to the operations team. The devops movement tries to address these issues on a technical and organisational level.

Emergencies

Another common question is "We're not able to keep WIP limits because of emergencies that pop up from time to time". In a well balanced flow, all stages in the Kanban flow are close to their WIP limits. This means that when an emergency pops up, the team won't be able to start

solving it until new capacity becomes available. This is not acceptable because an emergency can't wait for half a day before someone starts working on it.

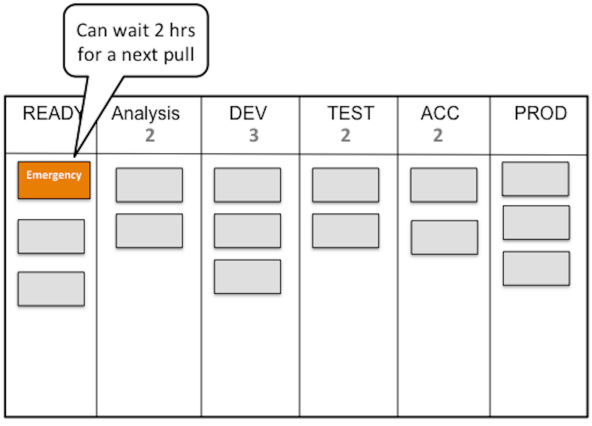


An emergency that can't wait

Reality proves to be different, even in Kanban land. When an emergency arrives, it needs to be dealt with, immediately! This means WIP limits can be broken. However, the decision needs to be made very consciously because of the impact. Introducing an emergency into the flow, causes people to drop what they're doing and switch context. This comes with a cost. The flow will be interrupted and lead times will be impacted. You probably understand that it's important to reduce these interruptions to a minimum. Otherwise variability of the Kanban flow

will increase, tearing down its reliability and frequent delivery of value. It doesn't take long before you fall back into estimating and planning.

One way to reduce the number of emergencies that interrupt the flow, is by making sure new items are pulled from the queue on a regular basis. When on average, it takes only 2 hours for a feature at the top of the queue to get pulled, far fewer emergencies need to be pushed into the flow. When you can rely on the pull frequency of the Kanban flow, it's much easier to just prioritize it in the queue instead of interrupting every body's work.



An emergency that can wait to be pulled

Off course there will always be emergencies that need to be dealt with immediately. This is where the concept

of Classes of Service[1] can offer you some flexibility. By categorizing the type of work that enters the Kanban flow, we agree to the service levels that apply to these different categories. Some categories have much higher due date performance than others. This gives you some room to absorb variation because of emergencies that disrupt the flow.

Many Kanban implementations use an expedite lane to deal with emergencies. This is a horizontal lane that spreads across all workflow stages and has a WIP limit of it's own. When an emergency arrives, it is pulled through the expedite lane with priority over all other work items.

4.3 Collaboration - daily stand-up

A common practice in agile teams is the daily stand-up. Each day, the team gathers in front of the task board and plans its next day. What have we worked on, what are we going to do, and are there any impediments? The daily stand-up is adopted by many Kanban teams, and proves to add a lot of value. By gathering around the Kanban board, the team gets the opportunity to inspect the flow and help each other to keep it going.

This is often the time when the team makes decisions on how to prevent upcoming bottlenecks. It doesn't take a genius to see where the flow will get obstructed. The team will often feel in advance when WIP limits will be

[1] http://www.amazon.com/Kanban-Successful-Evolutionary-Technology-Business/dp/0984521402

reached. By synchronizing on a regular basis they can make decisions on how to help each other and reduce the number of bottlenecks that cause people to be idle. For instance, when the development sees that nothing is moving in the next stage, which often is testing, they will raise this in the daily standup as a concern. Why? Because they know it will impact them later. This works downstream as well as upstream. A bottleneck will make it impossible for downstream stages to pull new items, and prevent upstream stages to pull new items because WIP limits are reached due to the bottleneck stage not freeing up capacity by pulling its items.

In a waterfall process, the development team doesn't care about the work of the test team. If they can't keep up, that's not their problem. The development team can just start working on the next feature. This is much harder in Kanban due to the WIP limits. The egocentric focus is reduced by the simple introduction of WIP limits. It doesn't matter how many features you can finish, if the next stage in the flow is not able to follow.

4.4 Summary

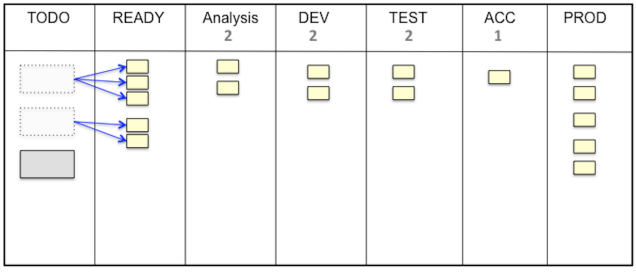
Off course introducing WIP limits is going to cause problems! The question is, why is that bothering you? The purpose of WIP limits is to make it easier to work in a pull system that is more effective from start to end. The WIP limits will signal an uneven balance in your flow. From that point, you can start working on improving it.

5 Stakeholders don't care about feeding the flow

Ordering the input queue is what determines which features get delivered to our customers, and which aren't. One of the most important aspects of Kanban is to make good decisions when it comes to prioritization. A common argument against Kanban is that stakeholders will never want to order the input queue on a regular basis. They don't want to get involved so closely since their time is precious and they can't commit to being available when needed. Even worse, stakeholders in general want all the features as fast as possible, so why bother ordering the list?

5.1 Ordering triggers business value

How many projects have you been part of, where half of the features had no clear definition of why they should be delivered? Sadly, this is sometimes the case even for the project itself. Stakeholders prioritizing the input queue, doesn't necessarily mean they are working on a feature level. In most cases it's on a higher level like a cluster of features that provide a cohesive set of added value to their customers.



Stakeholders prioritize big requests that get split up

In order to get the most out of your Kanban flow, it must be clear which rules apply when prioritizing features. These rules will lead to a definition of ready for features waiting to be ordered in the input queue. Without the necessary data, it is impossible to determine which priority a new feature will get in the input queue. For stakeholders this is a major incentive to do their homework. Their features won't get a place on the input queue, unless they can provide the necessary data. Kanban doesn't prescribe which data to collect for a feature. There are several interesting questions to ask depending on the organizational context.

- What is the type of feature? (new, bug, enhancement, ...)
- What is the business value?

- What is the cost of delay and which type? (insert reference)
- Any dependencies on other features?
- ...

These are all important factors to consider when determining the order of the input queue. They will help your stakeholders manage scope in a much more aggressive manner, allowing much less work on useless functionality that used to sneak into projects from all directions. This is important for both, the cost and value, two things all stakeholders are extremely interested in.

5.2 Building an MVP

The one thing stakeholders care most about is return on investment. What will this new product or feature bring to the company? Will we attract more customers? Will it help customer retention? Does it increase revenue and/or profit? How quickly will we reach our sales forecast?

These will all remain questions as long as the product hasn't shipped. In this day and age, these questions are major risks. The sooner we can get an answer, the better. The last thing we want is to spend months of work and budget on something that doesn't deliver what we initially thought it would.

This is where the concept of a Minimum Viable Product (MVP) can offer a solution.

An MVP is not a minimal product, it is a strategy for

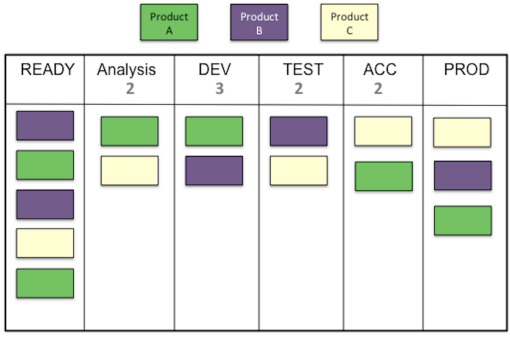
creating and selling a product to customers. It is an iterative learning process until the product fits the market, or is cancelled. The Lean Startup[1] movement offers great advice and examples on how to approach this. If it's applicable to your type of business, check it out!

Kanban makes it easier to build a MVP in the true sense of the term. We've known MVP's for a long time in software development under the term 1 version. However this is pursued as a minimal viable product, it often contains much more features than needed, due to scope creep and an upfront selection of features that continues to be chased even when some of them might no longer be the best choice. The single piece ordering of the input queue makes it easier to prevent this from happening. Stakeholders get a chance to critically review the scope on a continuous basis and apply the learning of a system in development. In many cases, experiencing delivered features gives them new ideas and insights, resulting in a better understanding of what the MVP should contain. That's when the real learning can begin, when you release a version to the public. However, the barrier to draw a line for a first release is easier because no major investments have been made upfront to detail out the features that are waiting in the queue.

[1] http://www.amazon.com/The-Lean-Startup-Entrepreneurs-Continuous/dp/0307887898/

5.3 Stakeholder collaboration

As a stakeholder you're seldom in a position where you get to use the full capacity of a product development team. Capacity is often shared amongst different stakeholders, often with something in common. A team can be working on multiple products or modules at a time. In that case, you need to agree with other stakeholders on the order of the input queue because it contains requests from different stakeholders.



A Kanban flow serving 3 products

In the past, when people had to work on different projects at the same time, this created a complex planning issue. People needed to be scheduled in advance to projects that were in the pipeline, keeping into account

their estimated free capacity. Due to unforeseen delays or extensions, this proved to be very inadequate and people were often transferred between projects to get one certain specialist on the new project. It was mostly the project manager who had to negotiate with department managers to get the people he needed. Politics were the key to success in this game, as you can imagine. You give some, you take some, leading to other projects losing important team members due to reassignments.

Kanban simplifies this issue significantly. By letting a team work on one continuous flow, they don't need to be reassigned over and over again. The focus has shifted to the input queue. Remember that the order of the features in the input queue determines what the team will be working on, in the near future. To simplify it, we have one factory that works on an order list for multiple clients. This implies that team members need to be multi disciplinary to a certain degree. They need to be able to work at different products, one request after another, while they used to specialize in one domain. Of course there needs to be a good balance, where you have a good mix of generalists and specialists. A strong argument to tilt this balance towards multi discipline is flexibility. When one specialist falls sick for a month, others can jump in and secure the continuation of the work.

If stakeholders can focus on the product, and not on resource planning, they can make better economical decisions for the company. While politics used to determine which project got priority, it is now a business decision, based on the needs of the customers and the organization.

Stakeholders can persuade each other why their features need to be high on the input queue because of the rules we all agreed on and the data that has been collected to prove it. Do you feel the shift from a silo focus towards an organizational and customer focus? It may sound naive to think that senior management will trust each other and stop playing the political games. That's why it's important that company leadership give full support and makes sure these rules are followed by all. Just like in a factory, capacity is a given, and all stakeholders have a right to a piece of it. Software development being an intangible business makes it much more needed to set clear rules to avoid abuse.

5.4 Expectation management

Stakeholders need regular status updates from the teams working on their requests. In many organizations this is done by steering committees on a monthly basis, where project managers of different teams elaborate on progress of their teams. These are meetings with a high political content. More than half the time of these meetings is spent on trying to get extra capacity or priority over other projects. Simply because stakeholders know the game. If they don't do it, others will steal capacity from them.

The rules on the Kanban input queue completely change these status meetings. The focus changes to the features moving through the flow, instead of the fight to get as many people as possible. Expectations are clear from the

start. Your product gets 20% of the flow capacity, which you can see each day on the Kanban board with your own eyes, together with the capacity other products are using. It is a very transparent way of tracking progress. The beauty of this approach is that it will create trust, which is one of the hardest things to achieve on a senior management level. By using simple rules and complete transparency, suspiciousness will start to disappear. This creates an opening for stakeholders to work more cooperatively. When one product has a need for extra capacity, another stakeholder will much more quickly offer a piece of his capacity because he knows the system is treating every stockholder fairly, and he can do the same thing later.

Expectations towards release scheduling is much more comforting. While stakeholders used to get progress updates during steering committee meetings where no real commitments to release dates could be given, they can now rely on the Kanban flow and their allocated capacity. By monitoring the measurements on a regular basis, they get a good idea about the release schedule which is based on real data instead of status updates that are given in a politically charged meeting.

5.5 Summary

Stakeholders might not care about prioritizing the features in the input queue, but they certainly care about their capacity of the Kanban flow. The transparency of the Kanban flow is attractive for stakeholders who are suspicious by

nature. The information they used to get second hand is now visible on the Kanban board and offers them a great tool to focus on economic decisions instead of company politics. By collaborating with other stakeholders and working on a great feature composition, they can return to what's most important, creating customer value. When something as trivial as setting priorities can change the focus of senior management proves that Kanban is indeed a powerful change management approach.

6 We will lose team cohesion

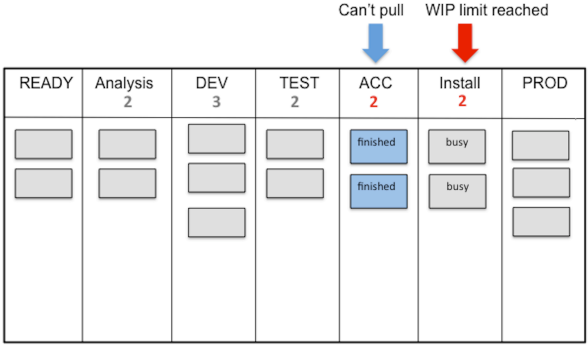
This argument comes in many cases from a project manager or a team leader. They have heard that there are no limits to the size of a Kanban team and are worried that the teams they have worked so hard to create, will fall apart. People will basically become factory line workers, who's only job is to keep up with the conveyor belt. How are we supposed to keep them motivated in this environment?

If there is one thing we learned over the years it's that the key to project success is communication. With teams of more than 10 people, we will lose the power of close collaboration and need to move back to more formal knowledge sharing through documentation. What about multi-disciplinary teams? By creating a team of business, technical and support roles, people got a chance to look across their borders and learn about the entire organization. If we divide them into separate specialities with the Kanban flow, won't we lose this?

6.1 Tearing down walls

As soon as a team gets involved in a Kanban flow, their cooperation mode starts to change. While they used to collaborate mostly inside the safety of their own team, they are suddenly depending on many other roles in the organization. A pull system with WIP limits makes dependencies on upstream and downstream stages more visible. When

your tasks is finished, you can only pull a new one when there is free capacity. When the WIP limit is reached, no more tasks can be pulled.



The Install stage can't pull new work due to a downstream bottleneck

Situations like these automatically lead to cross boundary communication. The person who is stuck due to another stage's WIP limit will go and check out what the problem is. Slowly but surely, the people who used to work in isolated teams will get to know the others who are also involved in the development of the end product.

This leads to another type of team feeling. One that includes all roles in the organization who are involved in the Kanban flow. IT teams in many organizations are still perceived as islands on their own, which is why business

analysts are introduced to bridge the gap between business and IT, a workaround to the actual problem.

By introducing Kanban, isolated teams fall apart and bigger teams start to emerge. Teams that contain a diversity of roles and personalities, each with their own knowledge to spread. You don't lose team cohesion, you get a new team composition that will build bridges in the organization.

6.2 Finding a bigger purpose

One of the prerequisites for building a great team is to find a common purpose. The team needs to be able to commit to the same goal which is the target for them to reach together. In vertically organized organizations, where silo's hand over work to each other, the purpose is internally oriented. The common purpose is strongly aligned with the goals of the department. IT cares about delivering what is requested by the different business units. The business units care about delivering what is requested by management. Sales cares about reaching sales targets, and so on…

The problem about these purposes is that they are not necessarily aligned. Sales might be doing a great job selling according to their targets, but not taking into account the capability of the other silo's that are involved in delivering the orders. The answer to mediate this mismatch is to create cross silo functions who's task lies in creating alignment and controlling the end to end process.

This is were Kanban strikes a chord. Teams get busted out of their silo thinking by becoming part of an end to end process where measurements and targets are set from customer request to product delivery. The focus on flow and continuous improvement will slowly but surely raise awareness of a bigger purpose, namely customer value. No matter how efficient you work, if the other parties involved in the system are not following along, little value is created. The visual aspect of Kanban plays an important part in this awareness. Many Kanban teams gather around the Kanban board on a daily basis to synchronize. How are tasks moving through the flow? Are there any bottlenecks present, and what can we do to exploit them?

From a change management perspective this has proven to be a very effective approach. People are naturally skeptical about new processes or changes in their work habits. Who can blame them? Most of us can't count the number of change initiatives they've been through on both hands. Many of them never resulted in anything substantial.

Kanban doesn't enforce any theoretical frameworks, training initiatives or redefinition of job descriptions. The job content of most people will stay exactly the same, only applied within a certain set of rules. Generally, there's very limited resistance in a Kanban adoption, which makes it a better candidate for sustainable change.

6.3 Creative thinking

In order for a team to grow, they need room to be creative and build solutions on their own, so they can take pride in their work. Without it, the team will lose motivation and start acting as a machine which will only produce according to what it is designed to create.

Agile methods have this aspect well covered. Instead of specifying each detail of the system to be implemented, each role interacts with the others in order to create a better solution together. No matter how hard we wish, people will always have certain competences in which they are better than others. But even the most senior architect will have a hard time to design the perfect system. So instead of trying to create this design to perfection up front, he will do this cooperatively with the team. This creates major opportunities for teams to be creative and share knowledge, two of the top ten most important job satisfaction criteria.

While Kanban doesn't prescribe this, its continuous flow stimulates strong collaboration. Remember the focus on reducing waste, which in many organizations comes in the form of outdated or misaligned requirement specifications. Kanban also focuses on improving lead and cycle time, which pushes the team to reduce the amount of isolated work and start with faster collaborative work. The focus on added value is the driver, where only the work that brings value to the customer is pursued.

When a team starts to investigate why the current flow is suboptimal, they often come to realize that it is because of a feedback loop that is too slow. An optimal flow has

limited returning cycles, where work is rejected because of a mismatch. This can be a bug, a missing piece of functionality, a missing scenario, … Either way, the feature can't flow further because something is missing. There are two ways to try and fix this. By expanding the degree of up front analysis and design, or by improving feedback loops. Because the first option has failed for centuries and will have a dramatic impact on our lead and cycle time, the second one is preferred. Better collaboration and faster feedback cycles will have a positive effect on flow and will not introduce extra waste into the flow.

All these effects of Kanban make it challenging and rewarding from a personal and team perspective. The team gets plenty of opportunities to be creative, collaborate and share knowledge on a continuous basis.

6.4 Participatory decision making

In a real team, decisions are made together. Each person's opinion is valued and heard. What happens when we move from a small specialized team to a larger Kanban team where much more roles are present?

This is an important concern, which needs to be addressed right from the start. When a Kanban team of sales, product management, development, testing, etc is assembled, there is a natural grasp for power, right from the start, where often the role with the most senior position in the organization will take control. This is a huge risk for the Kanban team as it will demotivate people and block

collaboration.

A Kanban team, like an agile team, needs to work together as one team. Otherwise they will just be a couple of sequenced specialists working a conveyor belt. Together they can reach so much more by starting the journey to improve flow and reduce waste by continuously improving their process.

This gap can be filled by a leader or coach, someone who will help the team on the process level. He will not be involved in the work, but help facilitate the team in their Kanban journey. His first task is to help the team to understand the principles of Kanban and learn what this means in their everyday job. This goes a long way, from working with product managers to system engineers. His second job is to facilitate participatory decision making. When the team gathers around the Kanban board or holds an improvement session, the coach helps the team to make decisions together. This is not an easy task, because even more people of different characters are present than in a typical specialized team.

The focus on reducing waste and improving flow helps to settle arguments. Even more effective is assessing measurements after improvement initiatives. This proves with hard data the effect of an improvement. Continuous improvement sessions can be held on a regular basis or on demand when the team notices need for improvement. These sessions are a great moment for a coach to help the team to grow in making decisions together and learn to understand why each opinion is valuable because in the end they all pursue the same goal which is to create value

to customers.

6.5 Achievable goals

One thing to keep motivation is to have goals, another thing is to have achievable goals. An illness of this age is the ever increasing pressure to deliver more and faster. We've all been part of projects where estimations where frowned upon and management pushed to commit to more.
In many cases, teams agree to lower their estimates, because they don't want to limit their career options in the company. Everybody knows this backfires pretty hard in the end. It is proven that people are bad at estimating, especially in knowledge work domains, such as software development. And we all know from experience that a sequence project plan gets delayed exponentially as soon as the slightest task gets overdue.

It doesn't take long for a team to get demotivated in this situation. Management keeps pushing the original deadline while the team knows there's no way they can ever live up to it without working 24/7. This is a killer for building great teams. Being on a death march like this will cause people to look for other opportunities, get sick, start the blame game, etc.

There's nothing wrong with ambitious goals, as long as they are achievable. This means the average peak of work, but with a general normal pace.

In Kanban teams don't start with the estimation fallacy, they start with a first version of their process flow and

collect measurements. This data gives an idea of the team capacity after a short period of time. Lead and cycle time averages give a clear idea about the real capabilities of the team in the current conditions. Management can pretty soon get an idea of the chances of success of their preferred release date. This doesn't mean the team cannot boost it's performance. A first version of their Kanban flow can be improved significantly. The big difference is that we're not pushing people to go faster, we're improving the conditions as a team, resulting in a more efficient end to end flow. The pace can remain the same, with occasional peaks instead of a continuous pressure which will burn out the team.

6.6 Summary

The last decades have had a strong emphasize on team building. We have all been on a day away from the office to do outdoor exercises with our colleagues. It appears that Kanban moves in the other direction, one with more hand-offs and less collaboration. This is a false perception, because it will create the opposite effect. People will tear down the walls between different stages in the flow. Simply because they focus on improving lead times. It gives people the opportunity to broaden their horizon, by helping other roles to get things flowing. Kanban, with it's strong focus on delivering end to end value, makes it easier for people to focus on what truly matters, a happy client. It will reduce silo thinking in the long run.

Teams will get plenty of opportunities to be creative.

They have the freedom to decide together on how they attack work and continuously improve their workflow. They are encouraged to do this together, since once change in the flow will impact all the others.

Finally, work will become realistic again because Kanban starts from reality by measuring instead of estimating. From that point, improvements can be made on the entire flow, without wasting time on isolated parts of the process.

7 Summary

I hope this book got you excited about Kanban and gave you insights why many arguments against it are invalid.

But beware that the difference between theory and practice is significant. Kanban will not suit every organizations or team. If you start from the fundamental question 'Why do we need a change management approach?', you will quickly get a feeling whether introducing Kanban will help the organization. When your organizational model doesn't align with the foundations of Kanban, you might want to consider other options first.

Nevertheless, Kanban is an exciting new approach worth studying and practicing. I expect it to gain even more traction in the combination with the Lean Startup idea's.

How lucky are we, to be working in a field that has so many new interesting evolutions?

8 Extras

My goal of this book was to provide clear answers to arguments against Kanban in software development. I started out with answering the 5 most common arguments, but soon enough I discovered more. In the next chapters you will find the arguments that didn't make it to the first cut, or new ones I documented after getting feedback from readers. Enjoy reading and don't hesitate to contact me if you have a new argument or wish to write a chapter yourself.

9 Software development is not manufacturing

This statement typically doesn't come from management, but from software developers. They know from experience that in software development, work is highly variable. Not many have developed the same system over and over again. Each new assignment requires them to start from scratch, on a technical as well as on a functional level. What kind of technology is appropriate? What is a proper architecture? What exactly do our end-users need?

Damn right if they are skeptical when management consultants compare software development with car manufacturing. It doesn't matter how successful Toyota has become with their Lean ideas, we're just not doing the same type of work! As soon as they have designed a car, it's mass production. Creating the same car over and over again. But prior to this, the company has spent a significant amount of money and time on R&D, car design and prototyping, building a plant, … Maybe that's why they call it lean manufacturing, because it only focuses on production…

If we would apply this to building a new software system, wouldn't we need to make the same distinction? Invest in a good preparation where you create an architecture, design and prototype the specification, create a solid development process. After it has been approved, start the mass production? But wait, we will only produce one system, right? So these lean principles are not applicable, because they are meant to optimize the production flow

of many identical products? Yes indeed, but that doesn't mean they are not applicable to software development.

9.1 Support teams

Why does Kanban feels like such a good fit for support teams? Because their situation comes closest to manufacturing. They already have a running system, which means their preparation phase has already passed and they're working in production mode. Most often, they are continuously working on requests in the form of bug fixes, improvements or new features. They are in fact pulling tasks from a prioritized input queue, just like a car assembly line is pulling new cars from a prioritized order list. Limiting WIP, enables a support team to remain responsive and deliver solutions to the users in a constant stream according to the service level agreement. They focus on efficiency and reliability.

In many organizations, support teams are managed as a capacity to process request, much like an assembly line is managed as a capacity to produce cars. Kanban feels like a next step to improve efficiency by optimizing capacity through evolutionary change caused by the introduction of a few basic rules. Limit work in progress, remove bottlenecks, visualize and continuously improve flow. Also effectiveness is increased by making process policies explicit. By studying demand and implementing proper classes of service, we can better meet our customers' expectations.

9.2 Development teams

In development teams Kanban feels completely different. These teams are designing new solutions, working on the edge of innovation, testing new concepts and changing accordingly. Their world is full of uncertainty which makes work unpredictable and highly variable. Planning always has been a difficult exercise for these teams.

Our natural reaction is to invest more in a solid preparation. Get the design right and then let a team develop the system based on rock-solid specifications. Just like designing a car and then managing it's mass production.

Many IT-professionals are still in denial, thinking they can design an entire new application without programming a single thing. Reality has proven that the gap between a useful application and its initial design doesn't shrink linearly when investing more in the upfront study. Instead, it's much more interesting to minimize the initial investment and use a feedback model that allows you to test ideas fast. This way, a team can validate all the initial hypotheses and learn what really matters to their customers.

A new car is not designed on paper only. BMW has recently revealed their car design process. As shown in the video[1], they rely heavily on creating prototypes for testing out ideas, many times in clay. From many different designs, in the end only one makes the final cut. It is a process of continuous learning, each step bringing them closer to the end product.

[1] http://www.bmw.com/com/en/insights/bmw_design/process.html

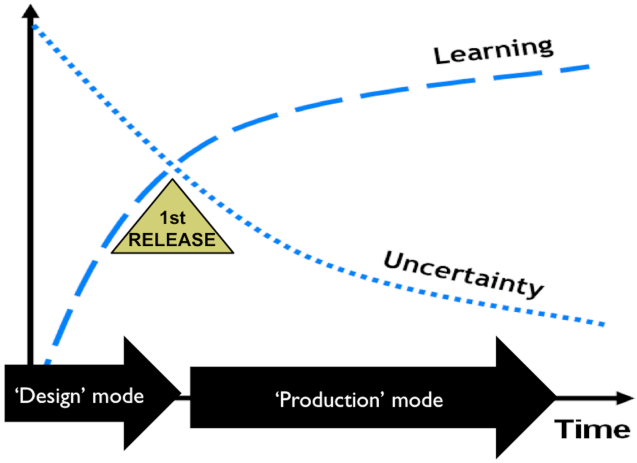
The same ideas can be applied to software development. Let's take 2 weeks to create a high level design of the new system, based on a vision. When this foundation is constructed, we will continuously test and learn by developing new pieces of the system. Learning leads to opportunities to correct our many assumptions. However, changing a system in development is hard, because many tasks are half finished and every piece of code seems to be connected. In Kanban, we try to make changes easier, by aggressively limiting the amount of half-finished work.

Won't this lead to strange results, like starting to build a bike and ending up with a snowboard?

Experience tells us that using a feedback model doesn't have such a dramatic impact on the product vision. It is within the boundaries of the bicycle vision that feedback cycles lead to changes in the bike design, which bring us closer to the user's needs. In the end, it is still a bicycle and you won't be able to comfortably descend a slope with it. Kanban in the most creative part of a project is less focused on efficiency, it helps a team to manage the complexity of change and learning by reducing work in progress and enabling fast feedback.

Most professionals have experienced the gradual evolution in a project many times, where at a certain point in time, variability starts to decrease and it feels like the project is in production mode. In many cases this happens after a first (minor) release. From that point on, planning starts to become controllable. Teams can really start to tune their process and it is typically in these periods that continuous improvement leads to better performance.

Kanban, with its focus on managing flow, makes it easier for teams to see patterns and take their cooperation to the next level.



Evolution towards production mode

In a Lean process, the team only works on customer request in their quest to minimize waste and increase added value for the customer. This seems crazy in a software development project, but it's not when you're following a continuous feedback model. A backlog initially only contains features that have been brainstormed by the team. You can consider these to be customer requests, with the product manager being a customer. As soon as the Kanban flow starts running, feedback is collected frequently. This will result in changes to the backlog. New

features are added, existing features are modified or even removed. Soon enough, the backlog contains many items that can be linked to real customers, the ones that are part of your customer discovery phase. Sure, sometimes the team needs a technical feature, like changing to a new technology. But even those features have to be linked to a customer request. Why would you implement a technical component, if nobody experiences a benefit? Maybe your users need higher performance, or your organization needs better alignment with other products,... When you're not able to link a feature to a user, start questioning its purpose.

9.3 Summary

Software development is not like manufacturing, that's for sure! But the ideas of Lean manufacturing are very applicable. Kanban has translated those ideas to the world of software development and is a good match if you want to:

- Manage the creative, discovery part of your project, by enabling fast feedback and the ability to realize the product vision by learning.
- Improve the repetitive, production part of your project, without sacrificing fast response.

10 Bibliography

Anderson, David J. Kanban. Sequim, WA: Blue Hole Press, 2010.

Cockburn, Alistair. Agile Software Development: The Cooperative Game (2nd Edition). Boston, MA: Addison-Wesley, 2006.

Imai, Masaaki. Gemba Kaizen: A Commonsense, Low-Cost Approach to Management. NY: McGraw-Hill, 1997.

Goldratt, Eliyahu M. & Cox, Jeff. The Goal: A Process of Ongoing Improvement. Boston, MA: North River Press, 2004.

Liker, Jeffrey. The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer. NY: McGraw-Hill, 2004.

Pink, Daniel H. Drive: The Surprising Truth About What Motivates Us. NY: Riverhead Books, 2009.

Poppendieck, Mary & Poppendieck Tom. Implementing Lean Software Development: From Concept to Cash. Boston, MA: Addison-Wesley, 2006.

Reinertsen, Donald G. Managing the Design Factory. NY: The Free Press, 1997.

Ries, Eric. The Lean Startup: How Today's Entrepreneurs User Continuous Innovation to Create Radically Successful Businesses. NY: Crown Business, 2011.

Seddon, John. Freedom from Command and Control. UK: Vanguard Consulting Ltd, 2003.

Womack, James P. & Jones, Daniel T. Lean Thinking: Banish Waste and Create Wealth in Your Corporation. NY: Free Press, 2003.