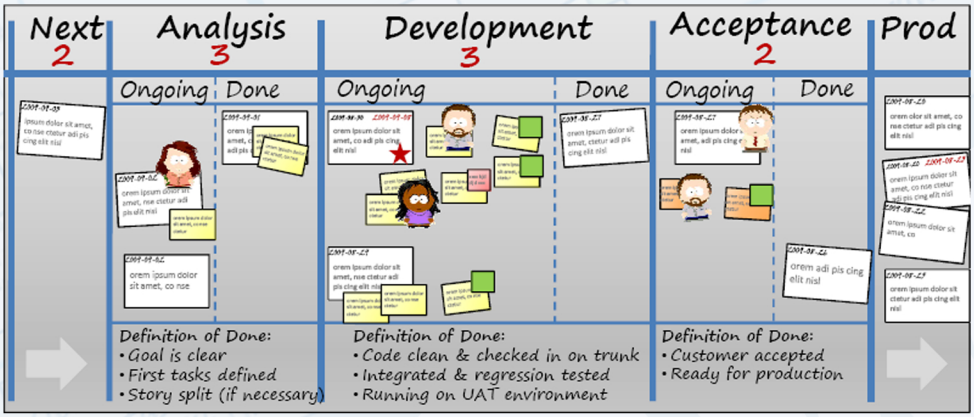


Kanban and Scrum
making the most of both

Henrik Kniberg & Mattias Skarin

Forewords by Mary Poppendieck and David Anderson

ENTERPRISE SOFTWARE
DEVELOPMENT SERIES

InfoQ queue

FREE ONLINE EDITION

If you like the book, please support
the authors and InfoQ by
purchasing the printed book:
http://www.lulu.com/content/7731694
(only $22.95)

Brought to you
Courtesy of

This book is distributed for free on InfoQ.com, if you have received this book from any other source then please support the authors and the publisher by registering on InfoQ.com.

Visit the homepage for this book at:

http://www.infoq.com/minibooks/kanban-scrum-minibook

Kanban and Scrum - making the most of both

Henrik Kniberg & Mattias Skarin
Forewords by Mary Poppendieck & David Anderson

Free Online Version

Support this work, buy the print copy:
http://www.infoq.com/minibooks/kanban-scrum-minibook

C4Media, Publisher of InfoQ.com.

This book is part of the InfoQ Enterprise Software Development series of books.

For information or ordering of this or other InfoQ books, please contact books@c4media.com.

Managing Editor: Diana Plesa
Cover art: Bistrian IOSIP
Composition: Accurance

Library of Congress Cataloguing-in-Publication Data:
ISBN: 978-0-557-13832-6

Printed in the United States of America

Contents

FOREWORD BY MARY POPPENDIECK ..... **v**
FOREWORD BY DAVID ANDERSON ..... **vii**
INTRODUCTION ..... **xi**
PART I – COMPARISON **1**
1. What is Scrum and Kanban anyway? 3
2. How do Scrum and Kanban relate to each other? 7
3. Scrum prescribes roles 11
4. Scrum prescribes timeboxed iterations 13
5. Kanban limits WIP per workflow state, Scrum limits WIP per iteration 15
6. Both are empirical 17
7. Scrum resists change within an iteration 23
8. Scrum board is reset between each iteration 25
9. Scrum prescribes cross-functional teams 27
10. Scrum backlog items must fit in a sprint 29
11. Scrum prescribes estimation and velocity 31
12. Both allow working on multiple products simultaneously 33
13. Both are Lean and Agile 35
14. Minor differences 37
15. Scrum board vs Kanban board – a less trivial example 41
16. Summary of Scrum vs Kanban 49
PART II – CASE STUDY **53**
17. The nature of technical operations 55
18. Why on earth change? 57
19. Where do we start? 59
20. Getting going 61
21. Starting up the teams 63
22. Addressing stakeholders 65
23. Constructing the first board 67
24. Setting the first Work In Progress (WIP) limit 71

25. Honoring the Work In Progress (WIP) limit 73
26. Which tasks get on the board? 75
27. How to estimate? 77
28. So how did we work, really? 79
29. Finding a planning concept that worked 83
30. What to measure? 87
31. How things started to change 91
32. General lessons learned 97
FINAL TAKE-AWAY POINTS 101
ABOUT THE AUTHORS 103

Foreword by Mary Poppendieck

Henrik Kniberg is one of those rare people who can extract the essence of a complicated situation, sort out the core ideas from the incidental distractions, and provide a crystal clear explanation that is incredibly easy to understand. In this book, Henrik does a brilliant job of explaining the difference between Scrum and Kanban. He makes it clear that these are just tools, and what you really want to do is have a full toolkit, understand the strengths and limitations of each tool and how to use them all.

In this book you will learn what Kanban is all about, its strengths and limitations, and when to use it. You will also get a good lesson on how and when to improve upon Scrum, or any other tool you may happen to be using. Henrik makes it clear that the important thing is not the tool you start with, but the way you constantly improve your use of that tool and expand your toolset over time.

The second part of this book by Mattias Skarin makes the book even more effective by walking you through the application of Scrum and Kanban in a real life situation. Here you will see an example of how the tools were used both separately and in combination to improve a software development process. You will notice that there isn't a single "best" way to do things; you have to think for yourself and figure out – based on your situation – your next step toward better software development.

Mary Poppendieck

Foreword by David Anderson

Kanban is based on a very simple idea. Work In Progress (WIP) should be limited and something new should be started only when an existing piece of work is delivered or pulled by a downstream function. The kanban (or signal card) implies that a visual signal is produced to indicated that new work can be pulled because current work does not equal the agreed limit. This doesn't sound very revolutionary nor does it sound like it would profoundly affect the performance, culture, capability and maturity of a team and its surrounding organization. What's amazing is that it does! Kanban seems like such a small change and yet it changes everything about a business.

What we've come to realize about Kanban is that it is an approach to change management. It isn't a software development or project management lifecycle or process. Kanban is an approach to introducing change to an existing software development lifecycle or project management methodology. The principle of Kanban is that you start with whatever you are doing now. You understand your current process by mapping the value stream and then you agree to WIP limits for each stage in that process. You then start to flow work through the system by pulling it when kanban signals are generated.

Kanban is proving useful to teams doing Agile software development but equally it is gaining traction with teams taking a more traditional approach. Kanban is being introduced as part of a Lean initiative to morph the culture of organizations and encourage continuous improvement.

Because WIP is limited in a Kanban system, anything that becomes blocked for any reason tends to clog up the system. If enough work items become blocked the whole process grinds to a halt. This has the effect of focusing the whole team and the wider organization on solving the problem, unblocking the item and restoring flow.

Kanban uses a visual control mechanism to track work as it flows through the various stages of the value stream. Typically, a whiteboard with sticky notes, or an electronic card wall system, is used. The best practice is probably to do both. The transparency that this generates also contributes to cultural change. Agile methods have been good about providing transparency into the WIP, completed work and reporting metrics such as velocity (the quantity of work done in an iteration). However, Kanban goes a step further and provides transparency into the process and its flow. Kanban exposes bottlenecks, queues, variability and waste – all of which are things which impact the performance of the organization in terms of the quantity of valuable work delivered and the cycle time required to deliver it. Kanban provides team members and external stakeholders with visibility into the effect of their actions (or inactions.) As such, early case studies are showing that Kanban changes behavior and encourages greater collaboration within the workplace. The visibility into and impact on bottlenecks, waste and variability also encourages discussion about improvements, and teams quickly start implementing improvements to their process.

As a result, Kanban encourages incremental evolution of existing processes and evolution that is generally aligned with Agile and Lean values. Kanban does not ask for a sweeping revolution of how people work, rather it encourages gradual change. It's change that is understood and agreed by consensus amongst the workers and their collaborators.

Through the nature of the pull system, Kanban also encourages delayed commitment on both prioritization of new work and delivery of existing work. Typically, teams will agree to a prioritization cadence for meeting with upstream stakeholders and decide what to work on next. These meetings can be held often because they are usually very short. A very simple question has to be answered, e.g. "Since our last meeting two slots have become free. Our current cycle time is 6 weeks to delivery. Which 2 things would you most like delivered 6 weeks from now?" This has a two-fold affect: asking a simple question generally in a quickly-derived, high-quality answer and also keeps the meeting short. The nature of question means that commitment on what to work on is delayed until the last responsible moment. This improves agility by managing expectations, shortening cycle times from commitment to delivery and eliminating rework since the chance that priorities will change is minimized.

One final word on Kanban is that the effect of limiting WIP provides predictability of cycle time and makes deliverables more reliable. The "stop the line" approach to impediments and bugs also appears to encourage very high levels of quality and a rapid drop in rework.

While all of this will become evident from the wonderfully clear explanations in this book, how we got here will remain opaque. Kanban was not conceived in a single afternoon through some incredible epiphany - instead it emerged over several years. Many of the profound psychological and sociological effects that change the culture, capability and maturity of organizations were never imagined. Rather, they were discovered. Many of the results with Kanban are counter-intuitive. What appears to be a very mechanical approach – limit WIP and pull work – actually has profound effects on people and how they interact and collaborate with one another. I, nor anyone else involved with Kanban in the early days, anticipated this.

I pursued what became Kanban as an approach to change which would meet with minimal resistance. This was clear to me as early as 2003. I also pursued it for the mechanical benefits. As I was discovering through application of Lean techniques around that time, if managing WIP made sense, then limiting it made more sense: it took the management overhead out of managing it. So in 2004, I decided to try implementing a pull system from first principles. I got the opportunity when a manager at Microsoft approached me and asked me to help him manage change on his team doing maintenance upgrades on internal IT applications. The first implementation was based on the Theory of Constraints pull system solution known as Drum-Buffer-Rope. It was a huge success: cycle time dropped by 92%; throughput increased by more than 3 times; and predictability (due date performance) was a very acceptable 98%.

In 2005, Donald Reinertsen persuaded me to implement a full-blown Kanban system. I got the opportunity in 2006 when I took charge of the software engineering department at Corbis in Seattle. In 2007, I began to report the results. The first presentation was at the Lean New Product Development Summit in Chicago in May 2007. I followed this with an open space at Agile 2007 in Washington DC in August that year. 25 people attended and 3 of them were from Yahoo!: Aaron Sanders, Karl Scotland and Joe Arnold. They went home to California, India and the UK

and implemented Kanban with their teams, which were already struggling with Scrum. They also started a Yahoo! discussion group which, at the time of this writing, has almost 800 members. Kanban was beginning to spread and early adopters were talking about their experiences.

Now in 2009, Kanban is really growing in adoption and more and more field reports are coming in. We've learned a lot about Kanban in the past 5 years and we all continue to learn more every day. I've focused my own work on doing Kanban, writing about Kanban, speaking about Kanban and thinking about Kanban in order to better understand it and explain it to others. I've deliberately stepped back from comparing Kanban to existing Agile methods, though some effort was spent in 2008 explaining why Kanban deserved to be considered an Agile-compatible approach.

I've left it to others with a wider experience to answer questions like "How does Kanban compare to Scrum?" I am so glad that Henrik Kniberg and Mattias Skarin have emerged as leaders in this regard. You, the knowledge worker in the field, need information in order to make informed decisions and move forward with your work. Henrik and Mattias are serving your needs in a way that I never could. I am particularly impressed with Henrik's thoughtful approach to comparison and his factual and un-opinionated, balanced delivery. His cartoons and illustrations are particularly insightful and often save you reading many pages of text. Mattias' field case study is important because it demonstrates that Kanban is much more than theory and it shows you by example how it might be useful for you in your organization.

I hope that you enjoy this book comparing Kanban with Scrum and that it gives you greater insight into Agile in general and both Kanban and Scrum in particular. If you would like to learn more about Kanban please visit our community website, The Limited WIP Society, http://www.limitedwipsociety.org/

David J. Anderson

Sequim, Washington, USA
July 8th, 2009

Introduction

We don't normally write books. We prefer spending our time deep in the trenches helping clients optimize, debug, and refactor their development process and organization. We've noticed a clear trend lately, though, and would like to share some thoughts on that. Here's a typical case:

- **Jim:** "Now we've finally gone all-out Scrum!"
- **Fred:** "So how's it going?"
- **Jim:** "Well, it's a lot better than what we had before..."
- **Fred:** "...but?"
- **Jim:** "... but you see we are a support & maintenance team."
- **Fred:** "yes, and?"
- **Jim:** "Well, we love the whole thing about sorting priorities in a product backlog, self-organizing teams, daily scrums, retrospectives, etc...."
- **Fred:** "So what's the problem?"
- **Jim:** "We keep failing our sprints"
- **Fred:** "Why?"
- **Jim:** "Because we find it hard to commit to a 2 week plan. Iterations don't make to much sense to us, we just work on whatever is most urgent for today. Should we do 1 week iterations perhaps?"
- **Fred:** "Could you commit to 1 week of work? Will you be allowed to focus and work in peace for 1 week?"

- **Jim:** "Not really, we get issues popping up on a daily basis. Maybe if we did 1 day sprints..."
- **Fred:** "Do your issues take less than a day to fix?"
- **Jim:** "No, they sometimes take several days"
- **Fred:** "So 1-day sprints wouldn't work either. Have you considered ditching sprints entirely?"
- **Jim:** "Well, frankly, we would like that. But isn't that against Scrum?"
- **Fred:** "Scrum is just a tool. You choose when and how to use it. Don't be a slave to it!"
- **Jim:** "So what should we do then?"
- **Fred:** "Have you heard of Kanban?"
- **Jim:** "What's that? What's the difference between that and Scrum?"
- **Fred:** "Here, read this book!"
- **Jim:** "But I really like the rest of Scrum though, do I have to switch now?"
- **Fred:** "No, you can combine the techniques!"
- **Jim:** "What? How?"
- **Fred:** "Just read on..."

Purpose of this book

If you're interested in agile software development you've probably heard about Scrum, and you may also have heard about Kanban. A question that we hear more and more often is "so what is Kanban, and how does it compare to Scrum?" Where do they complement each other? Are there any potential conflicts?

The purpose of this book is to clear up the fog, so you can figure out how Kanban and Scrum might be useful in your environment.

Let us know if we succeed!

Part I – Comparison

This first part of the book is an attempt to make an objective and practical comparison of Scrum and Kanban. It is a slightly updated version of the original article "Kanban vs. Scrum" from April 2009. That article became popular, so I decided to turn it into a book and ask my colleague Mattias to spice it up with a "from the trenches" case study from one of our clients. Great stuff! Feel free to skip ahead to Part II if you prefer starting with the case study, I won't be offended.

Well, maybe just a little.

/Henrik Kniberg

Free Online Version

Support this work, buy the print copy:
http://www.infoq.com/minibooks/kanban-scrum-minibook

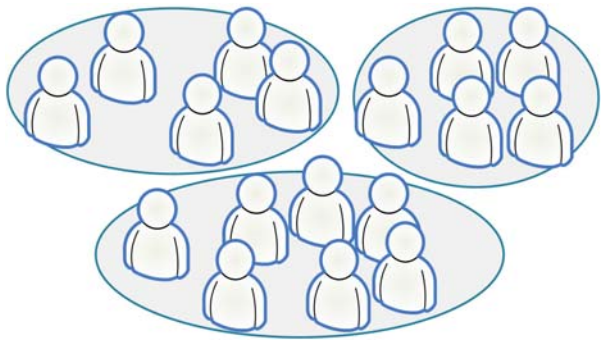
1

What are Scrum and Kanban anyway?

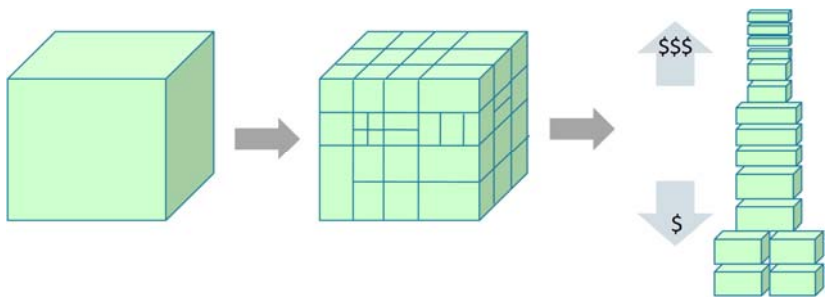
OK let's try to summarize Scrum and Kanban in less than 100 words each.

Scrum in a nutshell

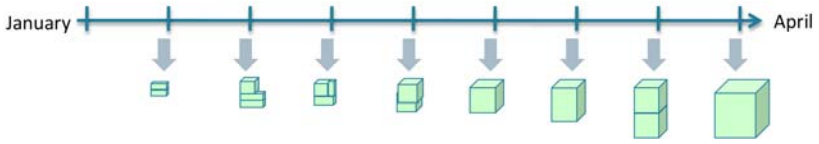
- **Split your organization** into small, cross-functional, self-organizing teams.



- **Split your work** into a list of small, concrete deliverables. Sort the list by priority and estimate the relative effort of each item.



- **Split time** into short fixed-length iterations (usually 1 – 4 weeks), with potentially shippable code demonstrated after each iteration.



- **Optimize the release plan** and update priorities in collaboration with the customer, based on insights gained by inspecting the release after each iteration.

- **Optimize the process** by having a retrospective after each iteration.

So instead of a **large group** spending **a long time** building a **big thing**, we have a **small team** spending a **short time** building a **small thing**. But **integrating regularly** to see the whole.

121 words... close enough.

For more details check out “Scrum and XP from the Trenches”. The book is a free read online. I know the author, he’s a nice guy :o)

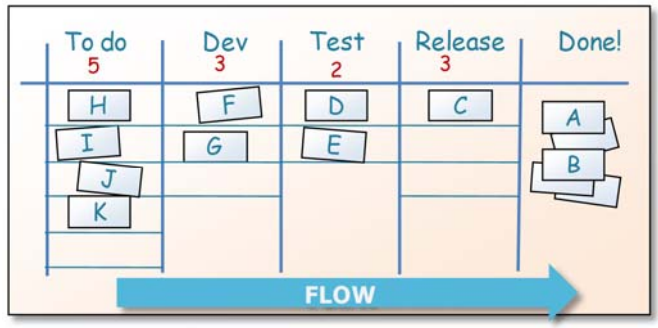
http://www.crisp.se/ScrumAndXpFromTheTrenches.html

For more Scrum links check out http://www.crisp.se/scrum

Kanban in a nutshell

- **Visualize the workflow**
    - Split the work into pieces, write each item on a card and put on the wall.
    - Use named columns to illustrate where each item is in the workflow.
- **Limit Work In Progress (WIP)** – assign explicit limits to how many items may be in progress at each workflow state.

- **Measure the lead time** (average time to complete one item, sometimes called "cycle time"), optimize the process to make lead time as small and predictable as possible.



We collect useful Kanban links at: http://www.crisp.se/kanban

2

How do Scrum and Kanban relate to each other?

Scrum and Kanban are both process tools

Tool = anything used as a means of accomplishing a task or purpose. *Process* = how you work.

Scrum and Kanban are *process tools* in that they help you work more effectively by, to a certain extent, telling you what to do. Java is also a tool, it gives you a simpler way to program a computer. A toothbrush is a also a tool, it helps you reach your teeth so you can clean them.

Compare tools for understanding, not judgment

Knife or fork – which tool is better?

Pretty meaningless question right? Because the answer depends on your context. For eating meatballs the fork is probably best. For chopping mushrooms the knife is probably best. For drumming on the table either will do fine. For eating a steak you probably want to use both tools together. For eating rice... well... some prefer a fork while others prefer chopsticks.

So when we compare tools we should be careful. Compare for understanding, not for judgment.

No tool is complete, no tool is perfect

Like any tools, Scrum and Kanban are neither perfect nor complete. They don't tell you *everything* that you need to do, they just provide certain constraints & guidelines. For example, Scrum constrains you to have timeboxed iterations and cross-functional teams, and Kanban constrains you to use visible boards and limit the size of your queues.

Interestingly enough, the value of a tool is that it *limits your options*. A process tool that lets you do anything is not very useful. We might call that process "Do Whatever" or how about "Do The Right Thing". The "Do The Right Thing" process is guaranteed to work, it's a silver bullet! Because if it doesn't work, you obviously weren't following the process :o)

Using the right tools will help you succeed, but will not guarantee success. It's easy to confuse *project* success/failure with *tool* success/failure.

- A project may succeed because of a great tool.
- A project may succeed despite a lousy tool.
- A project may fail because of a lousy tool.
- A project may fail despite a great tool.

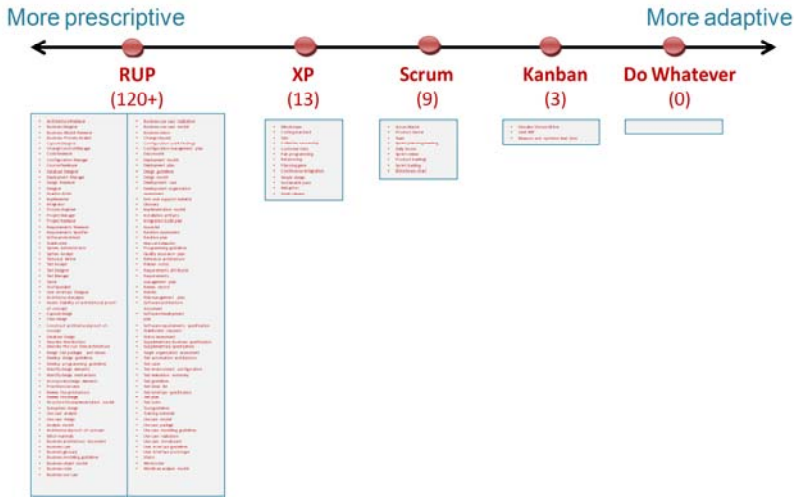
Scrum is more prescriptive than Kanban

We can compare tools by looking at how many rules they provide. *Prescriptive* means "more rules to follow" and *adaptive* means "fewer rules to follow". 100% prescriptive means you don't get to use your brain, there is a rule for everything. 100% adaptive means Do Whatever, there are no rules or constraints at all. As you can see, both extremes of the scale are kind of ridiculous.

Agile methods are sometimes called *lightweight* methods, specifically because they are less prescriptive than traditional methods. In fact, the first tenet of the Agile Manifesto is "Individuals and Interactions over Processes and Tools".

Scrum and Kanban are both highly adaptive, but *relatively speaking* Scrum is more prescriptive than Kanban. Scrum gives you more constraints, and thereby leaves fewer options open. For example Scrum prescribes the use of timeboxed iterations, Kanban doesn't.

Let's compare some more process tools on the prescriptive vs adaptive scale:



RUP is pretty prescriptive – it has over 30 roles, over 20 activities, and over 70 artifacts; an overwhelming amount of stuff to learn. You aren't really supposed to use all of that though; you are supposed to select a suitable subset for your project. Unfortunately this seems to be hard in practice. "Hmmmm... will we need *Configuration audit findings* artifacts? Will we need a *Change control manager* role? Not sure, so we better keep them just in case." This may be one of the reasons why RUP implementations often end up quite heavy-weight compared to Agile methods such as Scrum and XP.

XP (eXtreme Programming) is pretty prescriptive compared to Scrum. It includes most of Scrum + a bunch of fairly specific engineering practices such as test-driven development and pair programming.

Scrum is less prescriptive than XP, since it doesn't prescribe any specific engineering practices. Scrum is more prescriptive than Kanban though, since it prescribes things such as iterations and cross-functional teams.

One of the main differences between Scrum and RUP is that in RUP you get too much, and you are supposed to remove the stuff you don't need. In Scrum you get too little, and you are supposed to add the stuff that is missing.

Kanban leaves almost everything open. The only constraints are Visualize Your Workflow and Limit Your WIP. Just inches from Do Whatever, but still surprisingly powerful.

Don't limit yourself to one tool!

Mix and match the tools as you need! I can hardly imagine a successful Scrum team that doesn't include most elements of XP for example. Many Kanban teams use daily standup meetings (a Scrum practice). Some Scrum teams write some of their backlog items as Use Cases (a RUP practice) or limit their queue sizes (a Kanban practice). Whatever works for you.

Miyamoto Musashi a famous 17th century Samurai who was famous for his twin-sword fighting technique, said it nicely:

> Do not develop an attachment to any one weapon or any one school of fighting.
>
> - Miyamoto Musashi

Pay attention to the constraints of each tool though. For example if you use Scrum and decide to stop using timeboxed iterations (or any other core aspect of Scrum), then don't say you're using Scrum. Scrum is minimalistic enough as it is, if you remove stuff and still call it Scrum then the word will become meaningless and confusing. Call it something like "Scrum-inspired" or "a subset of Scrum" or how about "Scrumish" :o)

3

Scrum prescribes roles

Scrum prescribes 3 roles: Product Owner (sets product vision & priorities), Team (implements the product) and Scrum Master (removes impediments and provides process leadership).

Kanban doesn't prescribe any roles at all.

That doesn't mean you can't or shouldn't have a Product Owner role in Kanban! It just means you don't *have to*. In both Scrum and Kanban you are free to add whatever additional roles you need.

Be careful when adding roles though, make sure the additional roles actually add value and don't conflict with other elements of the process. Are you sure you need that Project Manager role? In a big project maybe that's a great idea, perhaps that's the guy who helps to synchronize multiple teams & product owners with one another. In a small project that role might be waste, or worse, might lead to sub-optimization and micromanagement.

The general mindset in both Scrum and Kanban is "less is more". So when in doubt, start with less.

In the rest of the book I'm going to use the term "Product Owner" to represent whoever it is that sets the priorities of a team, regardless of the process used.

4

Scrum prescribes timeboxed iterations

Scrum is based on timeboxed iterations. You can choose the length of the iteration, but the general idea is to keep the same length of iteration over a period of time and thereby establish a *cadence*.

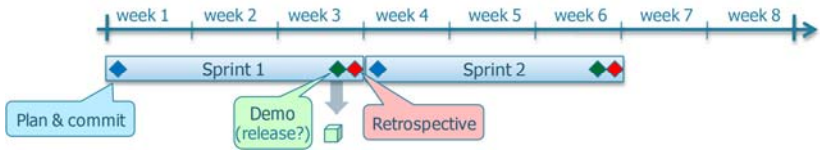
- **Beginning of iteration:** An iteration plan is created, i.e. team pulls out specific number items from the product backlog, based on the product owner's priorities and how much the team thinks they can complete in one iteration.
- **During iteration:** Team focuses on completing the items they committed to. The scope of the iteration is fixed.
- **End of iteration:** Team demonstrates working code to the relevant stakeholders, ideally this code should be *potentially shippable* (i.e. tested and ready to go). Then the team does a retrospective to discuss and improve their process.

So a Scrum iteration is one single timeboxed cadence combining three different activities: planning, process improvement, and (ideally) release.

In Kanban timeboxed iterations are not prescribed. You can choose when to do planning, process improvement, and release. You can choose to do these activities on a regular basis ("release every Monday") or on-demand ("release whenever we have something useful to release").

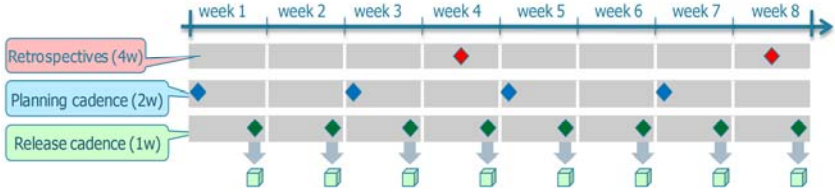
Team #1 (single cadence)

"We do Scrum iterations"



Team #2 (three cadences)

"We have three difference cadences. Every week we release whatever is ready for release. Every second week we have a planning meeting and update our priorities and release plans. Every fourth week we have a retrospective meeting to tweak and improve our process"



Team #3 (mostly event-driven)

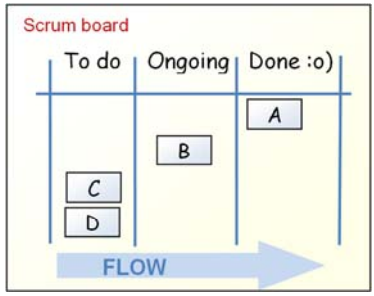
"We trigger a planning meeting whenever we start running out of stuff to do. We trigger a release whenever there is a set of Minimum Marketable Features (MMFs) ready for release. We trigger a spontaneous quality circle whenever we bump into the same problem the second time. We also do a more in-depth retrospective every fourth week."

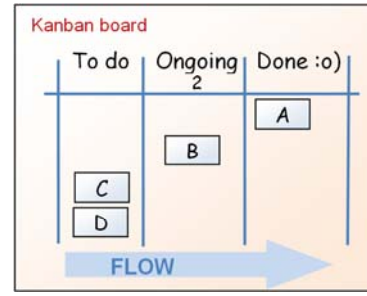
5

Kanban limits WIP per workflow state, Scrum limits WIP per iteration

In Scrum, the sprint backlog shows what tasks are to be executed during the current iteration (= "sprint" in Scrum-speak). This is commonly represented using cards on the wall, called a Scrum board or Task board.

So what's the difference between a Scrum board and a Kanban board? Let's start with a trivially simple project and compare the two:





In both cases we're tracking a bunch of items as the progress through a workflow. We've selected three states: To Do, Ongoing, and Done. You can choose whatever states you like – some teams add states such as Integrate, Test, Release, etc. Don't forget the *less is more* principle though.

So what's the difference between these two sample boards then? Yep - the little 2 in the middle column on the kanban board. That's all. That 2 means "there may be no more than 2 items in this column at any given moment".

In Scrum there is no rule preventing the team from putting all items into the Ongoing column at the same time! However there is an implicit limit since the iteration itself has a fixed scope. In this case the implicit limit per column is 4, since there are only 4 items on the whole board. So Scrum limits WIP indirectly, while Kanban limits WIP directly.

Most Scrum teams eventually learn that it is a bad idea to have too many ongoing items, and evolve a culture of trying to get the current items done before starting new items. Some even decide to explicitly limit the number of items allowed in the Ongoing column and then – tadaaa! – the Scrum board has become a Kanban board!

So both Scrum and Kanban limit WIP, but in different ways. Scrum teams usually measure *velocity* – how many items (or corresponding units such as "story points") get done per iteration. Once the team knows their velocity, that becomes their WIP limit (or at least a guideline). A team that has an average velocity of 10 will usually not pull in more than 10 items (or story points) to a sprint.

So in Scrum *WIP is limited per unit of time.*

In Kanban *WIP is limited per workflow state*.

In the above Kanban example, at most 2 items may be in the workflow state "Ongoing" at any given time, regardless of any cadence lengths. You need to choose what limit to apply to which workflow states, but the general idea is to limit WIP of *all* workflow states, starting as early as possible and ending as late as possible along the value stream. So in the example above we should consider adding a WIP limit to the "To do" state as well (or whatever you call your input queue). Once we have WIP limits in place we can start measuring and predicting lead time, i.e. the average time for an item to move all the way across the board. Having predictable lead times allows us to commit to SLAs (service-level agreements) and make realistic release plans.

If the item sizes vary dramatically then you might consider having WIP limits defined in terms of story points instead, or whatever unit of size you use. Some teams invest effort in breaking down items to roughly the same size to avoid these types of considerations and reduce time spent on estimating things (you might even consider estimation to be waste). It's easier to create a smooth-flowing system if items are roughly equal-sized.

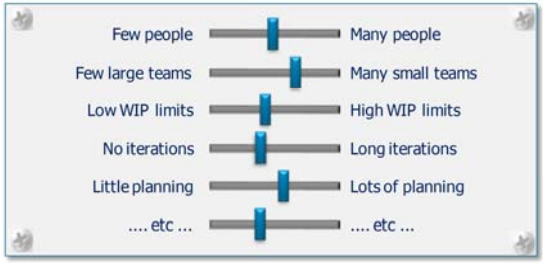
6

Both are empirical

Imagine if there were knobs on these meters, and you could configure your process by simply turning the knobs. "I want high capacity, low lead time, high quality, and high predictability. So I'll turn the knobs to 10, 1, 10, 10 respectively."

Wouldn't that be great? Unfortunately there are no such direct controls. Not that I know of at least. Let me know if you find any.

Instead what we have is a bunch of *indirect* controls.



Scrum and Kanban are both empirical in the sense that you are expected to experiment with the process and customize it to your environment. In fact, you *have to* experiment. Neither Scrum nor Kanban provide all the answers – they just give you a basic set of constraints to drive your own process improvement.

- Scrum says you should have cross-functional teams. So who should be on what team? Don't know, experiment.

- Scrum says the team chooses how much work to pull into a sprint. So how much should they pull in? Don't know, experiment.

- Kanban says you should limit WIP. So what should the limit be? Don't know, experiment.

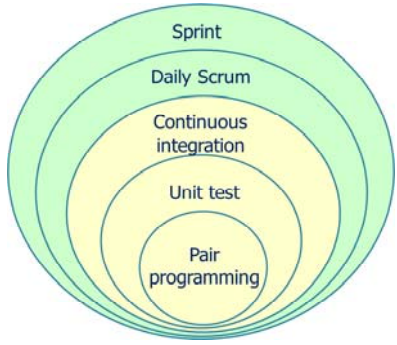
As I mentioned earlier, Kanban imposes fewer constraints than Scrum. This means you get more parameters to think about, more knobs to turn. That can be both a disadvantage and an advantage depending on your context. When you open up the configuration dialog of a software tool, do you prefer having 3 options to tweak, or 100 options to tweak? Probably somewhere in between. Depends on how much you need to tweak and how well you understand the tool.

So let's say we reduce a WIP limit, based on the hypothesis that this will improve our process. We then observe how things like capacity, lead time, quality, and predictability change. We draw conclusions from the results and then change some more things, continuously improving our process.

There are many names for this. Kaizen (continuous improvement in Lean-speak), Inspect & Adapt (Scrum-speak), Empirical Process Control, or why not The Scientific Method.

The most critical element of this is the *feedback loop*. Change something => Find out how it went => Learn from it => Change something again. Generally speaking you want as short a feedback loop as possible, so you can adapt your process quickly.

In Scrum, the basic feedback loop is the sprint. There are more, however, especially if you combine with XP (eXtreme programming):



When done correctly, Scrum + XP gives you a bunch of extremely valuable feedback loops.

The inner feedback loop, pair programming, is a feedback loop of a few seconds. Defects are found and fixed within seconds of creation ("Hey, isn't that variable supposed to be a 3?"). This is an "are we building the stuff **right**?" feedback cycle.

The outer feedback loop, the sprint, gives a feedback cycle of a few weeks. This is a "are we building the **right** stuff?" feedback cycle.

What about Kanban then? Well, first of all you can (and probably should) put all of the above feedback loops into your process whether or not you use Kanban. What Kanban then gives you is a few very useful real-time metrics:

- Average lead time. Updated every time an item reaches “Done” (or whatever you call your right-most column).
- Bottlenecks. Typical symptom is that Column X is crammed with items while column X+1 is empty. Look for “air bubbles” on your board.

The nice thing about real-time metrics is that you can choose the length of your feedback loop, based on how often you are willing to analyze the metrics and make changes. Too long feedback loop means your process improvement will be slow. Too short feedback loop means your process might not have time to stabilize between each change, which can cause thrashing.

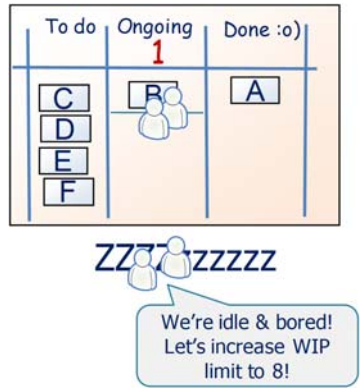
In fact, the length of the feedback loop itself is one of the things you can experiment with... sort of like a meta-feedback loop.

OK I’ll stop now.

Example: Experimenting with WIP limits in Kanban

One of the typical “tweak points” of Kanban is the WIP limit. So how do we know if we got it right?

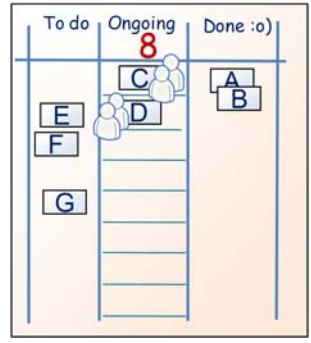
Let’s say we have a 4 person team, and we decide to start with a WIP limit of 1.



Whenever we start working on one item, we can't start any new item until the first item is Done. So it will get done really quickly.

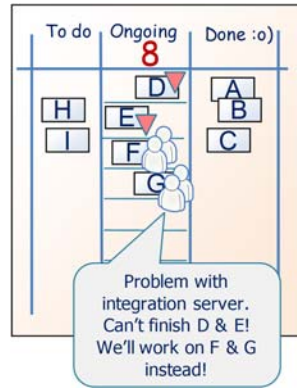
Great! But then it turns out that it's usually not feasible for all 4 people to work on the same item (in this sample context), so we have people sitting idle. If that only happens once in a while that's no problem, but if it happens regularly, the consequence is that the average lead time will increase. Basically, WIP of 1 means items will get through "Ongoing" really fast once they get in, but they will be stuck in "To Do" longer than necessary, so the total lead time across the whole workflow will be unnecessarily high.

So if WIP of 1 was too low, what about increasing it to 8?

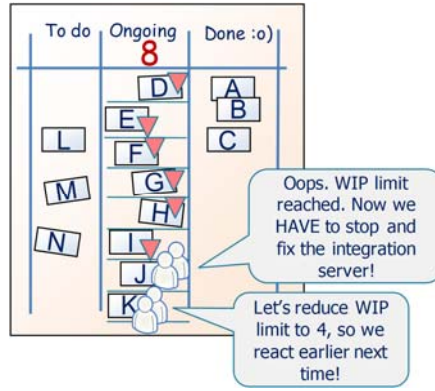


That works better for a while. We discover that, on average, working in pairs gets the work done most quickly. So with a 4 person team, we usually have 2 ongoing items at any given time. The WIP of 8 is just an upper limit, so having fewer items in progress is fine!

Imagine now, however, that we run into a problem with the integration server, so we can't fully complete any items (our definition of "Done" includes integration). That kind of stuff happens sometimes right?

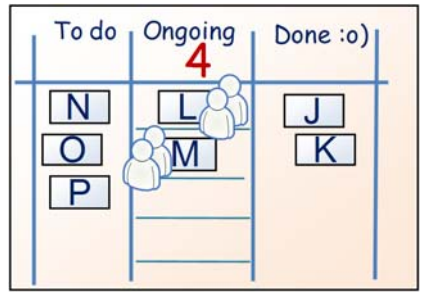


Since we can't complete item D or E, we start working on item F. We can't integrate that one either, so we pull in a new item G. After a while we hit our Kanban limit – 8 items in "Ongoing":



At that point we can't pull in any more items. Hey we better fix that danged integration server! The WIP limit has prompted us to react and fix the bottleneck instead of just piling up a whole bunch of unfinished work.

That's good. But if the WIP limit was 4 we would have reacted a lot earlier, thereby giving us a better average lead time. So it's a balance. We measure average lead time and keep optimizing our WIP limits to optimize the lead time:



After a while we might find that items pile up in “To do”. Maybe it’s time to add a WIP limit there as well then.

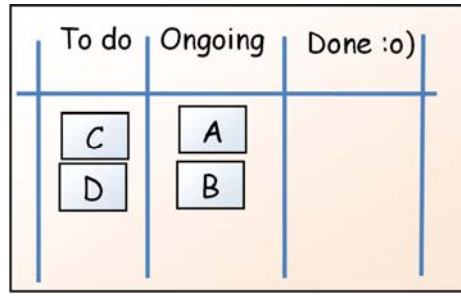
Why do we need a “To do” column anyway? Well, if the customer was always available to tell the team what to do next whenever they ask, then the “To do” column wouldn’t be needed. But in this case the customer is sometimes not available, so the “To Do” column gives the team a small buffer to pull work from in the meantime.

Experiment! Or, as the Scrumologists say, Inspect & Adapt!

7

Scrum resists change within an iteration

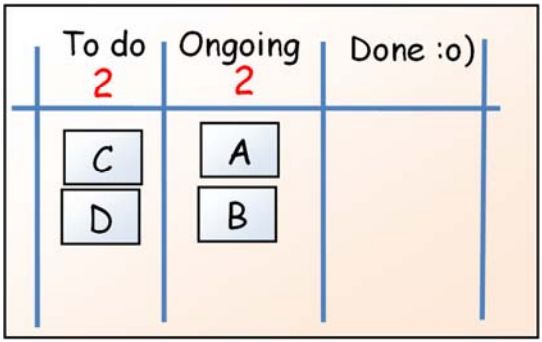
Let's say our Scrum board looks like this:



What if someone turns up and wants to add E to the board?

A Scrum team will typically say something like "No, sorry, we've committed to A+B+C+D this sprint. But feel free to add E to the product backlog. If the product owner considers it to be high priority we will pull this into next sprint." Sprints of the right length give the team just enough focused time to get something done, while still allowing the product owner to manage and update priorities on a regular basis.

So what would the Kanban team say then?



A Kanban might say “Feel free to add E to the To Do column. But the limit is 2 for that column, so you will need to remove C or D in that case. We are working on A and B right now, but as soon as we have capacity we will pull in the top item from To Do”.

So the response time (how long it takes to respond to a change of priorities) of a Kanban team is however long it takes for capacity to become available, following the general principle of “one item out = one item in” (driven by the WIP limits).

In Scrum, the response time is half the sprint length on average.

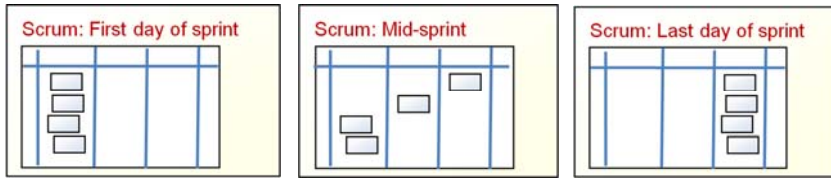
In Scrum, the product owner can’t touch the Scrum board since the team has committed to a specific set of items in the iteration. In Kanban you have to set your own ground rules for who is allowed to change what on the board. Typically the product owner is given some kind of “To Do” or “Ready” or “Backlog” or “Proposed” column to the far left, where he can make changes whenever he likes.

These two approaches aren’t exclusive though. A Scrum team *may* decide to allow a product owner to change priorities mid-sprint (although that would normally be considered an exception). And a Kanban team *may* decide to add restrictions about when priorities can be changed. A Kanban team may even decide to use timeboxed fixed-commitment iterations, just like Scrum.

8

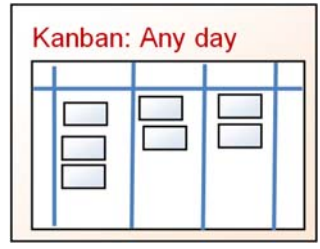
Scrum board is reset between each iteration

A Scrum board typically looks something like this during different stages of a sprint.



When the sprint is over, the board is cleared – all items are removed. A new sprint is started and after the sprint planning meeting we have a new Scrum board, with new items in the left-most column. Technically this is waste, but for experienced Scrum teams this usually doesn't take too long, and the process of resetting the board can give a nice sense of accomplishment and closure. Sort of like washing dishes after dinner – doing it is a pain but it feels nice afterwards.

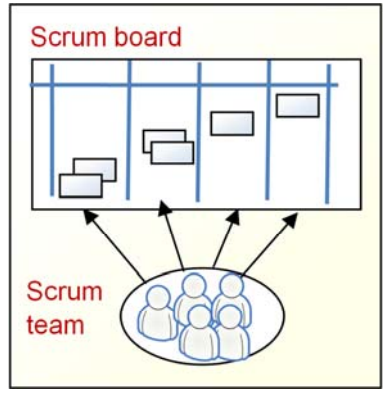
In Kanban, the board is normally a persistent thing – you don't need to reset it and start over.



9

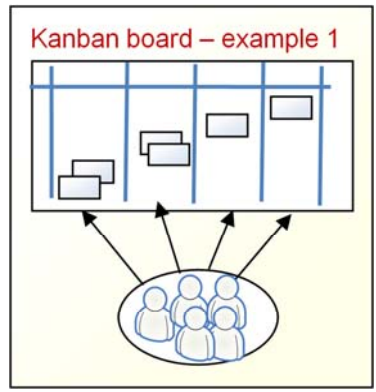
Scrum prescribes cross-functional teams

A Scrum board is owned by exactly one Scrum team. A Scrum team is cross-functional, it contains all the skills needed to complete all the items in the iteration. A Scrum board is usually visible to whoever is interested, but only the owning Scrum team may edit it – it is their tool to manage their commitment for this iteration.

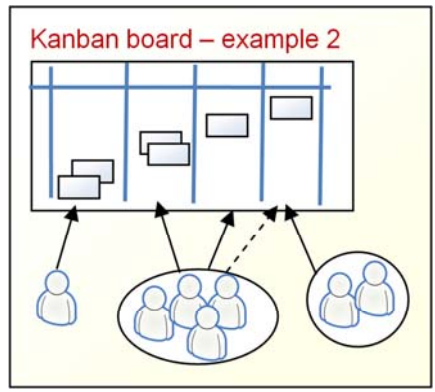


In Kanban, cross-functional teams are optional, and a board doesn't need to be owned by one specific team. A board is related to one workflow, not necessarily one team.

Here are two examples:



Example 1: The whole board is served by one cross-functional team. Just like Scrum.



Example 2: The product owner sets priorities in column 1. A cross-functional development team does development (column 2) and test (column 3). Release (column 4) is done by a specialist team. There is slight overlap in competencies, so if the release team becomes a bottleneck one of the developers will help them.

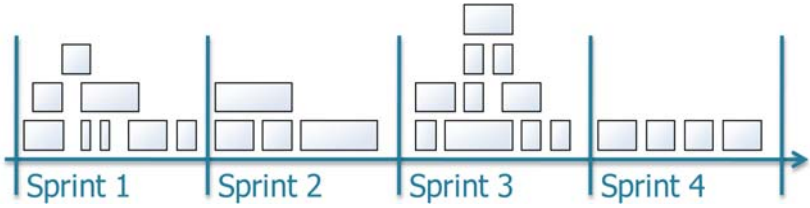
So in Kanban you need to establish some ground rules for who uses the board and how, then experiment with the rules to optimize flow.

10

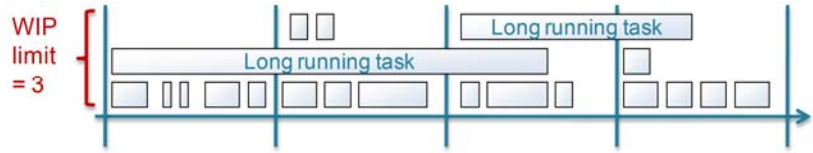
Scrum backlog items must fit in a sprint

Both Scrum and Kanban are based on incremental development, i.e. break the work into smaller pieces.

A Scrum team will only commit to items that they think they can complete within one iteration (based on the definition of "Done"). If an item is too big to fit in a sprint, the team and product owner will try to find ways to break it into smaller pieces until it does fit. If items tend to be big, iterations will be longer (although usually no longer than 4 weeks).



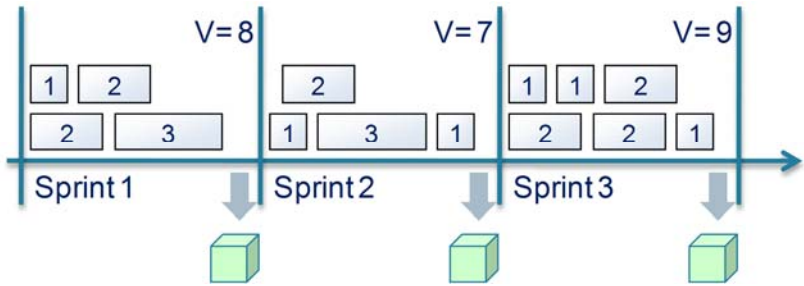
Kanban teams try to minimize lead time and level the flow, so that indirectly creates an incentive to break items into relatively small pieces. But there is no explicit rule stating that items must be small enough to fit into a specific time box. On the same board we might have one item that takes 1 month to complete and another item that takes 1 day.



11

Scrum prescribes estimation and velocity

In Scrum, teams are supposed to estimate the relative size (= amount of work) of each item that they commit to. By adding up the size of each item completed at the end of each sprint, we get velocity. Velocity is a measure of capacity – how much stuff we can deliver per sprint. Here's an example of a team with an average velocity of 8.



Knowing that the average velocity is 8 is nice, because then we can make realistic predictions about which items we can complete in upcoming sprints, and therefore make realistic release plans.

In Kanban, estimation is not prescribed. So if you need to make commitments you need to decide how to provide predictability.

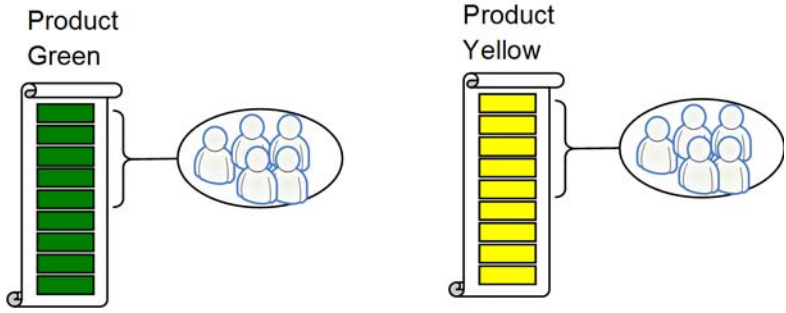
Some teams choose to make estimates and measure velocity just like in Scrum. Other teams choose to skip estimation, but try to break each item into pieces of roughly the same size – then they can measure velocity simply in terms of how many items were completed per unit of time (for example features per week). Some teams group items into MMFs and measure the average lead time per MMF, and use that to establish Service-Level Agreements (SLAs) – for example "when we commit to an MMF it will always be delivered within 15 days".

There’s all kinds of interesting techniques for Kanban-style release planning and commitment management – but no specific technique is prescribed so go ahead and Google away and try some different techniques until you find one that suits your context. We’ll probably see some “best practices” emerge over time.

12

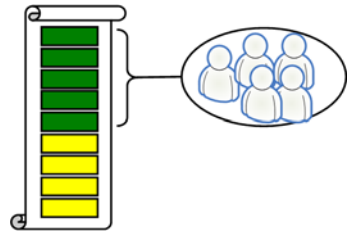
Both allow working on multiple products simultaneously

In Scrum, "Product Backlog" is a rather unfortunate name since it implies that all items have to be for the same product. Here are two products, green and yellow, each with their own product backlog and their own team:

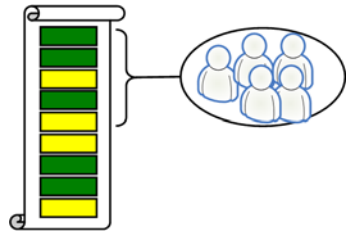


What if you only have one team then? Well, think of Product Backlog more like a Team Backlog. It lists the priorities for upcoming iterations for one particular team (or set of teams). So if that team is maintaining multiple products, merge both products into one list. That forces us to prioritize between the products, which is useful in some cases.

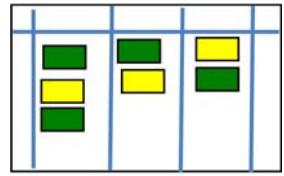
There are several ways of doing this in practice. One strategy would be to have the team focus on one product per sprint:



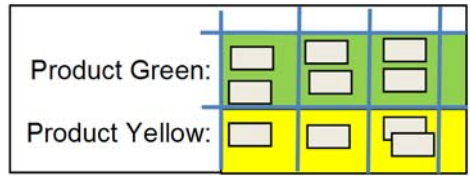
Another strategy would be to have the team work on features from both products each sprint:



It's the same in Kanban. We can have several products flowing across the same board. We might distinguish them using different colored cards:



... or by having "swimlanes" :



13

Both are Lean and Agile

I'm not going to go through Lean Thinking and the Agile Manifesto here, but generally speaking both Scrum and Kanban are well aligned with those values and principles. For example:

- Scrum and Kanban are both pull scheduling systems, which corresponds to the JIT (Just In Time) inventory management principle of Lean. This means that the team chooses when and how much work to commit to, they "pull" work when they are ready, rather than having it "pushed" in from the outside. Just like a printer pulls in the next page only when it is ready to print on it (although there is a small & limited batch of paper that it can pull from).

- Scrum and Kanban are based on continuous and empirical process optimization, which corresponds to the Kaizen principle of Lean.

- Scrum and Kanban emphasize responding to change over following a plan (although Kanban typically allows faster response than Scrum), one of the four values of the agile manifesto.

... and more.

From one perspective Scrum can be seen as not-so-lean because it prescribes batching items into timeboxed iterations. But that depends on the length of your iteration, and what you are comparing to. Compared to a more traditional process where we perhaps integrate and release something 2 – 4 times per year, a Scrum team producing shippable code every 2 weeks is extremely lean.

But then, if you keep making the iteration shorter and shorter you are essentially approaching Kanban. When you start talking about making the iteration shorter than 1 week you might consider ditching timeboxed iterations entirely.

I’ve said it before and I’ll keep saying it: Experiment until you find something that works for you! And then keep experimenting :o)

14

Minor differences

Here are some differences that seem to be less relevant compared to the other ones mentioned before. It's good to be aware of them though.

Scrum prescribes a prioritized product backlog

In Scrum, prioritization is always done by sorting the product backlog, and changes to priorities take effect in the next sprint (not the current sprint). In Kanban you can choose any prioritization scheme (or even none), and changes take effect as soon as capacity becomes available (rather than at fixed times). There may or may not be a product backlog, and it may or may not be prioritized.

In practice, this makes little difference. On a Kanban board the left-most column typically fulfills the same purpose as a Scrum product backlog. Whether or not the list is sorted by priority, the team needs some kind of decision rule for which items to pull first. Examples of decision rules:

- Always take the top item.
- Always take the oldest item (so each item has a timestamp).
- Take any item.
- Spend approximately 20% on maintenance items and 80% on new features.
- Split the team's capacity roughly evenly between product A and product B.
- Always take red items first, if there are any.

In Scrum, a product backlog can also be used in a Kanban-ish way. We can limit the size of it, and create decision rules for how it should be prioritized.

In Scrum, daily meetings are prescribed

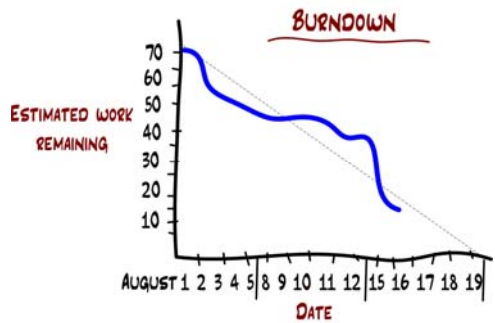
A Scrum team has a short meeting (at most 15 minutes) every day at the same time & same place. The purpose of the meeting is to spread information about what is going on, plan the current day's work, and identify any significant problems. This is sometimes called a daily standup, since it is usually done standing (to keep it short & maintain a high energy level).

Daily standups are not prescribed in Kanban, but most Kanban teams seem to do it anyway. It's a great technique, regardless of which process you use.

In Scrum the format of the meeting is people-oriented - every person reports one by one. Many Kanban teams use a more board-oriented format, focusing on bottlenecks and other visible problems. This approach is more scalable. If you have 4 teams sharing the same board and doing their daily standup meeting together, we might not necessarily have to hear everyone speak as long as we focus on the bottleneck parts of the board.

In Scrum, burndown charts are prescribed

A sprint burndown chart shows, on a daily basis, how much work remains in the current iteration.



The unit of the Y-axis is the same as the unit used on the sprint tasks. Typically hours or days (if the team breaks backlog items into tasks) or story points (if the team doesn't). There are lots of variations of this though.

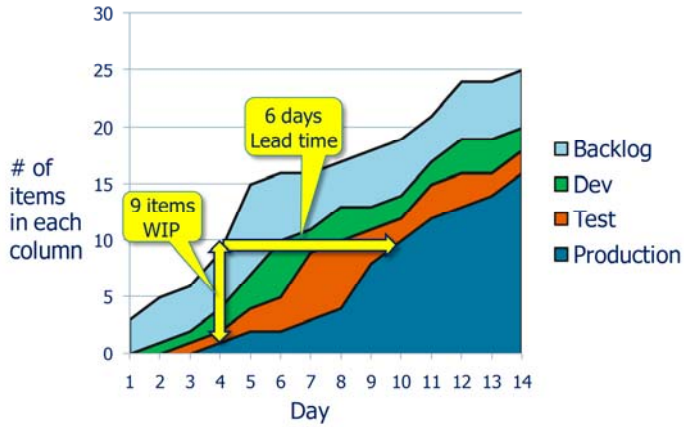
In Scrum, sprint burndown charts are used as one of the primary tools for tracking the progress of an iteration.

Some teams also use release burndown charts, which follows the same format but at a release level – it typically shows how many story points are left in the product backlog after each sprint.

The main purpose of a burndown chart is to easily find out as early as possible if we are behind or ahead of schedule, so that we can adapt.

In Kanban, burndown charts are not prescribed. In fact, no particular type of chart is prescribed. But you are of course allowed to use any type of chart you like (including burndowns).

Here's an example of a Cumulative Flow diagram. This type of chart illustrates nicely how smooth your flow is and how WIP affects your lead time.



Here's how it works. Every day, total up the number of items in each column on the Kanban board and stack these on the Y axis. So on day 4 there were 9 items in the board. Starting from the right-most column there was 1 item in Production, 1 item in Test, 2 items in Dev, and 5 items in the Backlog. If we plot these points every day and connect the dots we get a nice diagram like the one above. The vertical and horizontal arrows illustrate the relationship between WIP and lead time.

The horizontal arrow shows us that items added to the backlog on day 4 took on average 6 days to reach production. About half of that time was Test. We can see that if were to limit the WIP in Test and Backlog we would significantly reduce the total lead time.

The slope of the dark-blue area shows us the velocity (i.e. number of items deployed per day). Over time we can see how higher velocity reduces lead time, while higher WIP increases lead time.

Most organizations want to get stuff done faster (= reduce lead time). Unfortunately many fall into the trap of assuming that this means getting more people in or working overtime. Usually the most effective way to get stuff done faster is to smooth out the flow and limit work to capacity, *not* add more people or work harder. This type of diagram shows why, and thereby increases the likelihood that the team and management will collaborate effectively.

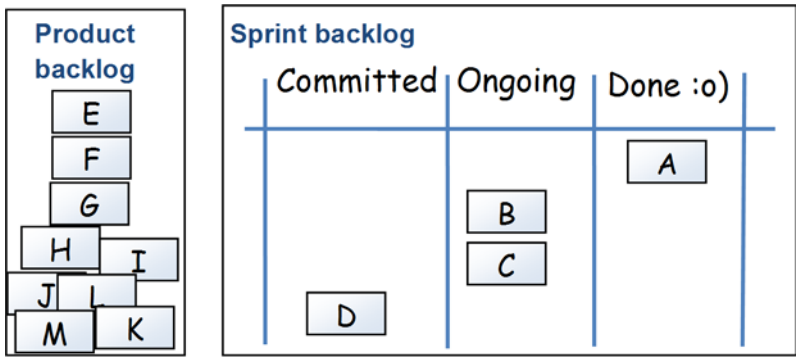
It gets even more clear if we distinguish between queuing states (such as "waiting for test") and working states (such as "testing"). We want to absolutely minimize the number of items sitting around in queues, and a cumulative flow diagram helps provide the right incentives for this.

15

Scrum board vs. Kanban board – a less trivial example

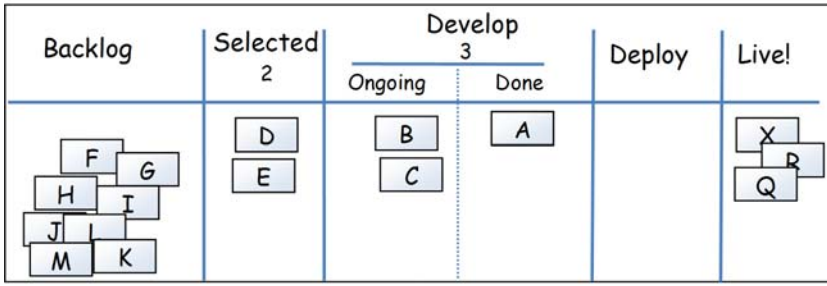
In Scrum, the sprint backlog is just one part of the picture – the part that shows what the team is doing during the current sprint. The other part is the product backlog – the list of stuff that the product owner wants to have done in future sprints.

The product owner can see but not touch the sprint backlog. He can change the product backlog any time he likes, but the changes don't take effect (i.e. change what work is being done) until next sprint.



When the sprint is done, the team "delivers potentially shippable code" to the product owner. So the team finishes the sprint, does a sprint review, and proudly demonstrates features A, B, C, and D to the product owner. The product owner can now decide whether or not to ship this. That last part – actually shipping the product – is usually not included in the sprint, and is therefore not visible in the sprint backlog.

Under this scenario, a Kanban board might instead look something like this:



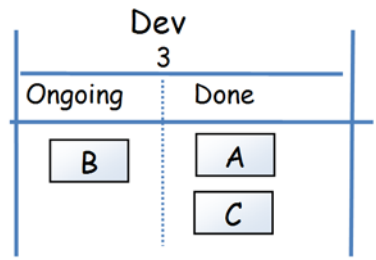
Now the whole workflow is on the same board – we're not just looking at what one Scrum team is doing in one iteration.

In the example above the "Backlog" column is just a general wish list, in no particular order. The "Selected" column contains the high priority items, with a Kanban limit of 2. So there may be only 2 high priority items at any given moment. Whenever the team is ready to start working on a new item, they will take the top item from "Selected". The product owner can make changes to the "Backlog" and "Selected" columns any time he likes, but not the other columns.

The "Dev" column (split into two sub-columns) shows what is current being developed, with a Kanban limit of 3. In network terms, the Kanban limit corresponds to "bandwidth" and lead time corresponds to "ping" (or response time).

Why have we split the "Dev" column into two sub-columns "Ongoing" and "Done"? That's to give the production team a chance to know which items they can pull into production.

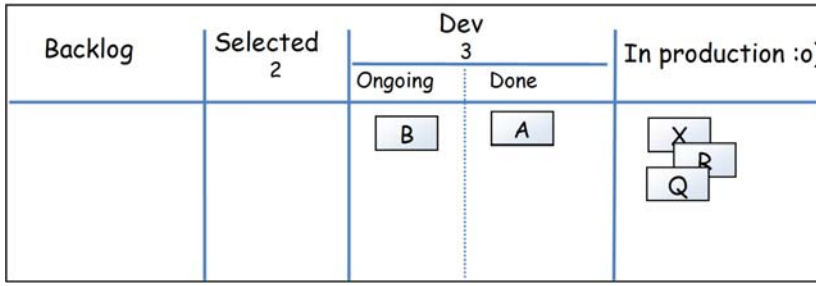
The "Dev" limit of 3 is shared among the two sub-columns. Why? Let's say there are 2 items in "Done":



That means there can only be 1 item in "Ongoing". That means there will be excess capacity, developers who *could* start a new item, but aren't allowed to because of the Kanban limit. That gives them a strong incentive to focus their efforts and helping to get stuff into production, to clear the "Done" column and maximize the flow. This effect is nice and gradual – the more stuff in "Done", the less stuff is allowed in "Ongoing" – which helps the team focus on the right things.

One-piece flow

One-piece flow is a kind of "perfect flow" scenario, where an item flows across the board without ever getting stuck in a queue. This means at every moment there is somebody working on that item. Here's how the board might look in that case:

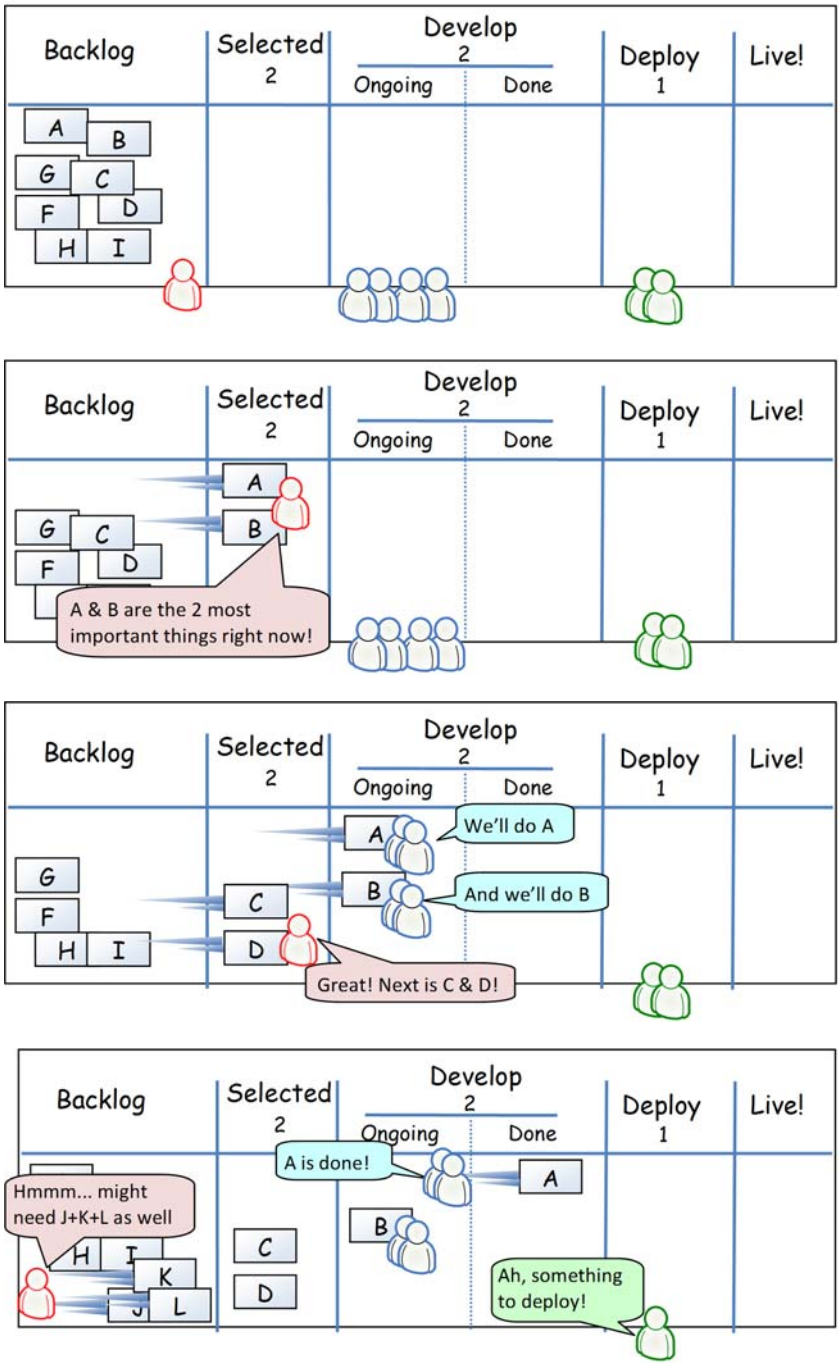


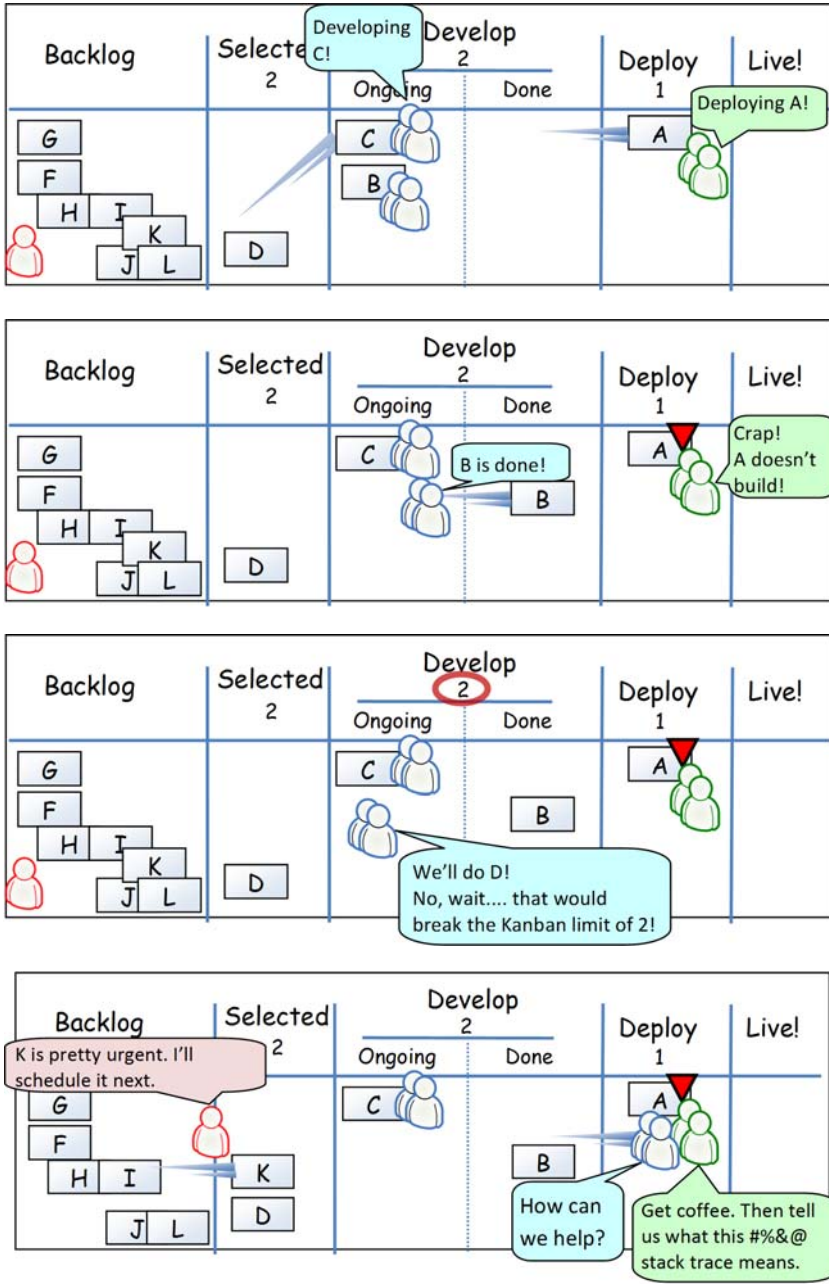
B is being developed at the moment, A is being put in production at the moment. Whenever the team is ready for the next item they ask the product owner that is most important, and get an instance response. If this ideal scenario persists we can get rid of the two queues "Backlog" and "Selected" and get a *really* short lead time!

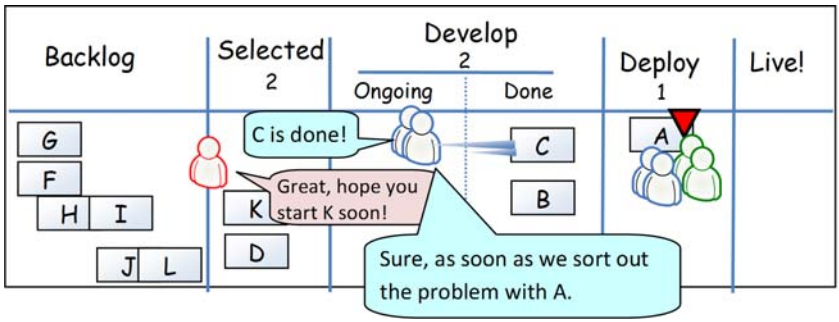
Cory Ladas puts it nicely: "The ideal work planning process should always provide the development team with best thing to work on next, no more and no less".

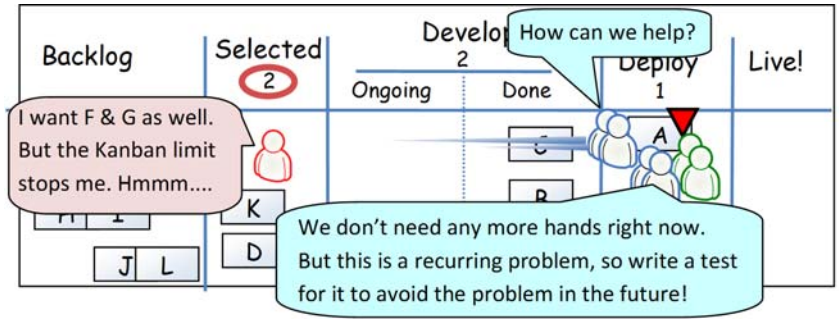
The WIP limits are there to stop problems from getting out of hand, so if things are flowing smoothly the WIP limits aren't really used.

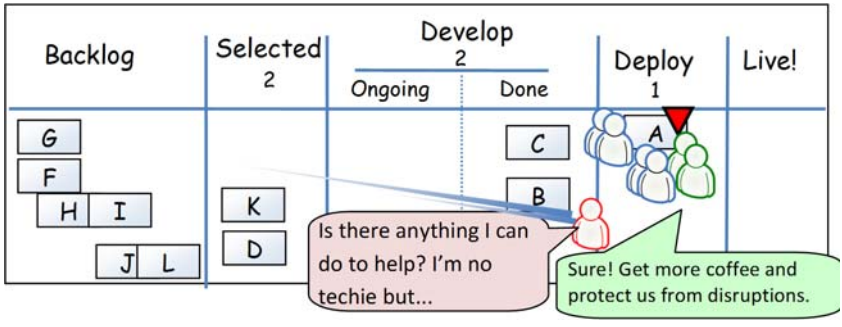
One day in Kanban-land

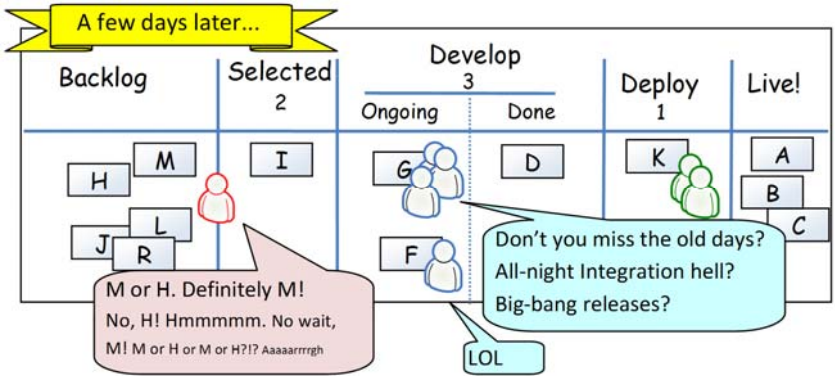



Backlog
Selecte
2
Developing C!
Develop
2
Ongoing
Done
Deploy
1
Live!
Deploying A!
G
F
H
I
K
J
L
D
C
B
A
Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
Crap! A doesn't build!
B is done!
G
F
H
I
K
J
L
D
C
B
A
Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
We'll do D! No, wait.... that would break the Kanban limit of 2!
G
F
H
I
K
J
L
D
C
B
A
Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
K is pretty urgent. I'll schedule it next.
How can we help?
Get coffee. Then tell us what this #%&@ stack trace means.
G
F
H
I
K
J
L
D
C
B
A


Backlog
Selected 2
Develop 2
Ongoing
Done
Deploy 1
Live!
G
F
H
I
J
L
K
D
C
B
A
C is done!
Great, hope you start K soon!
Sure, as soon as we sort out the problem with A.


Backlog
Selected 2
Develop 2
Ongoing
Done
Deploy 1
Live!
How can we help?
I want F & G as well. But the Kanban limit stops me. Hmmm....
K
D
J
L
A
We don't need any more hands right now. But this is a recurring problem, so write a test for it to avoid the problem in the future!


Backlog
Selected 2
Develop 2
Ongoing
Done
Deploy 1
Live!
G
F
H
I
J
L
K
D
C
B
A
Is there anything I can do to help? I'm no techie but...
Sure! Get more coffee and protect us from disruptions.


A few days later...
Backlog
Selected 2
Develop 3
Ongoing
Done
Deploy 1
Live!
H
M
I
J
L
R
G
D
F
K
A
B
C
M or H. Definitely M! No, H! Hmmmmm. No wait, M! M or H or M or H?!? Aaaaarrrrgh
Don't you miss the old days? All-night Integration hell? Big-bang releases?
LOL

Does the Kanban board have to look like this?

No, the board above was just an example!

The only thing that Kanban prescribes is that the work flow should be visual, and that WIP should be limited. The purpose is to create a smooth flow through the system and minimize lead time. So you need to regularly bring up questions such as:

Which columns should we have?
Each column represents one workflow state, or a queue (buffer) between two workflow states. Start simple and add columns as necessary.

What should the Kanban limits be?
When the Kanban limit for "your" column has been reached and you don't have anything to do, start looking for a bottleneck downstream (i.e. items piling up to the right on the board) and help fix the bottleneck. If there is no bottleneck that is an indication that the Kanban limit might be too low, since the reason for having the limit was to reduce the risk of feeding bottlenecks downstream.

If you notice that many items sit still for a long time without being worked on, that is an indication that the Kanban limit might be too high.

- Too low kanban limit => idle people => bad productivity
- Too high kanban limit => idle tasks => bad lead time

How strict are the Kanban limits?
Some teams treat them as strict rules (i.e. the team may not exceed a limit), some teams treat them as guidelines or discussion triggers (i.e. breaking a kanban limit is allowed, but should be an intentional decision with a concrete reason). So once again, it's up to you. I told you Kanban wasn't very prescriptive right?

16

Summary of Scrum vs. Kanban

Similarities

- Both are Lean and Agile.
- Both use pull scheduling.
- Both limit WIP.
- Both use transparency to drive process improvement.
- Both focus on delivering releasable software early and often.
- Both are based on self-organizing teams.
- Both require breaking the work into pieces.
- In both, the release plan is continuously optimized based on empirical data (velocity / lead time).

Differences

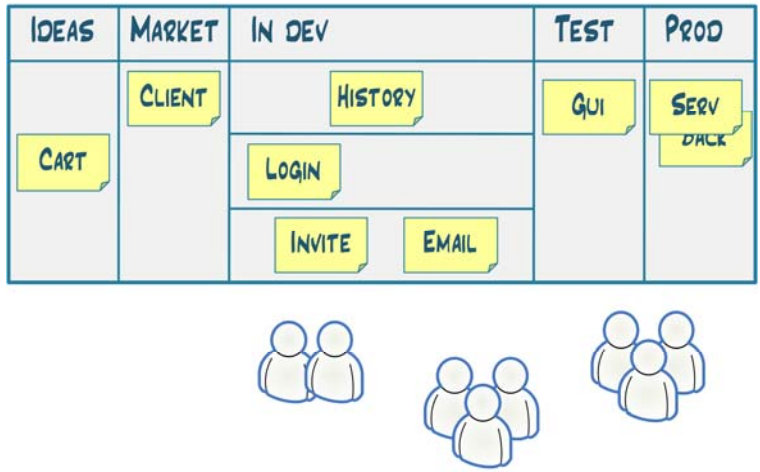
Scrum	Kanban
Timeboxed iterations prescribed.	**Timeboxed iterations optional.** Can have separate cadences for planning, release, and process improvement. Can be event-driven instead of timeboxed.
Team commits to a specific amount of work for this iteration.	**Commitment optional.**
Uses **Velocity** as default metric for planning and process improvement.	Uses **Lead time** as default metric for planning and process improvement.
Cross-functional teams prescribed.	Cross-functional teams optional. **Specialist teams allowed.**
Items must be broken down so they can be completed within 1 sprint.	No particular item size is prescribed.
Burndown chart prescribed	No particular type of diagram is prescribed
WIP limited indirectly (per sprint)	**WIP limited directly** (per workflow state)
Estimation prescribed	**Estimation optional**
Cannot add items to ongoing iteration.	**Can add new items whenever capacity is available**
A **sprint backlog is owned by one specific team**	A **kanban board may be shared by multiple teams** or individuals
Prescribes 3 roles (PO/SM/Team)	**Doesn't prescribe any roles**
A **Scrum board is reset** between each sprint	A **kanban board is persistent**
Prescribes a prioritized product backlog	**Prioritization is optional**.

There. That's it. Now you know the differences.

But it ain't over yet, now it's time for the best part! Put on your boots, it's time to jump into the trenches with Mattias and see what this looks like in practice!

Part II – Case study

Kanban in real life



This is a story of how we learned to improve using Kanban. When we started, not much information was around and Dr. Google for once left us empty-handed. Today, Kanban is evolving successfully and there is an emerging body of knowledge. I strongly recommend having a look at David Anderson's work, for example on 'classes of service'. So here comes the first (and last) disclaimer (promise!). Whatever solution you deploy, make sure they address your specific problems. There, done. Let's get onto it. So, this is our story.

/Mattias Skarin

Free Online Version

Support this work, buy the print copy:
http://www.infoq.com/minibooks/kanban-scrum-minibook

17

The nature of technical operations

If you ever have been on-call 24/7, you have a fair idea of the responsibility felt managing a production environment. You're expected to sort out the situation in the middle of the night regardless of whether or not you were the source of the problem. No one knows, that's why they call you. It is quite a challenge since you didn't create the hardware, drivers, OS or custom software. Your options are often limited to narrowing down the problem, limiting the impact, saving evidence needed to recreate the problem and waiting for the person responsible for causing the trouble to reproduce and solve the problem you just witnessed.

For technical operations, the speed and accuracy of both the responsiveness and the problem-solving are key.

18

Why on earth change?

In 2008 one of our clients, a Scandinavian game development organization, went through a whole series of process improvements. One of these included scaling Scrum to the development organization and piecemeal removal of impediments that were preventing development teams from delivering software. As software started flowing and performance increased, this magnified pressure downstream on technical operations. Previously the technical operations teams were mostly watching from the side, now they were becoming increasingly involved as an active party in the development process.

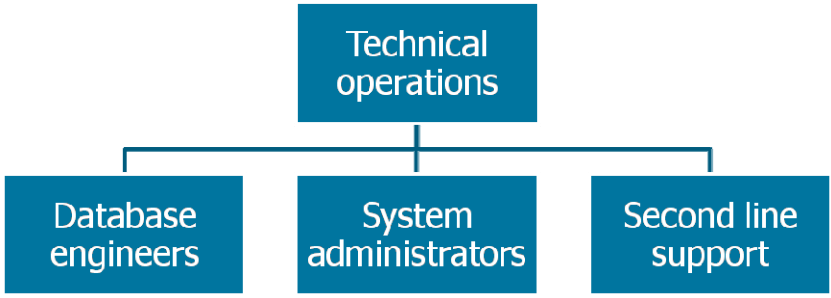


Figure 1. The organization of technical operations included three teams: the Database Administrators (DBAs), the system administrators and the second line support.

As a result, helping the development teams was not enough. If we kept focusing on only the development teams this would cause delays in critical infrastructure improvements run by the technical operations teams. Improvement in both areas was needed.

In addition, the progress in the development teams meant managers were increasingly requested to help out with analysis and feedback on ideas. This meant they had less time for real-time task prioritization and problem-solving. The management team realized that they needed to act before the situation became unmanageable.

19

Where do we start?

A good starting point was by asking the development teams, who were the customers of technical operations.

Development's view of operations

I asked "What are the top three things that come to mind when you think about 'operations'"? The most common answers were:

"Variable knowledge"

"Their workflow system sucks"

"Very competent when it comes to infrastructure"

"What are the guys doing?"

"They want to help, but actually getting help is hard"

"Needs many emails to do simple stuff"

"Projects take too long"

"Difficult to contact"

In brief, that was development's view of operations. Now, let's compare this with operations view of development..

Operations view of development

“Why aren’t you using the existing platform’s advantages?”

“Let’s make releases a less heavy affair!”

“We are hurt by your bad quality!”

“They ought to change” - was a common theme in the arguments from both sides. Obviously that mindset needed to change if we were to get traction in fixing common problems. On the positive side; “very competent when it comes to infrastructure” (indicating trust in core competence) made me believe that the “us vs. them” mentality could be fixed if we created the right work conditions. Eliminating overwork and focusing on quality was one viable option.

20

Getting going

So we needed to get going, but where do we start? The only thing we knew for certain: where we start won't be where we end up.

My background is that of a developer so I surely knew little about the nature of operations. I wasn't about to "storm in and start changing things". I needed a less confrontational approach that would still teach us the relevant things, discard the irrelevant things, and be easy to learn.

Candidates were:

1. Scrum – this was working well with the development teams.
2. Kanban – new & untested, but with good fit to the Lean principles that were lacking.

In discussions with the managers, Kanban and Lean principles seemed to match the problems we were trying to address. From their point of view sprints wouldn't fit very well, since they were doing re-prioritization on a daily basis. So Kanban was the logical starting point, even if it was a new animal for all of us.

21

Starting up the teams

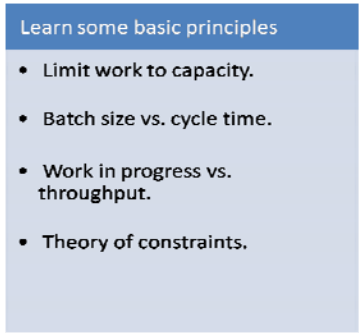
How do we start up the teams? There was no handbook out there on how to get going. Doing it wrong would be risking a lot. Apart from missing out on the improvements, we were we dealing with a production platform with highly specialized and skilled people which were hard to replace. Alienating them was a baaad idea.

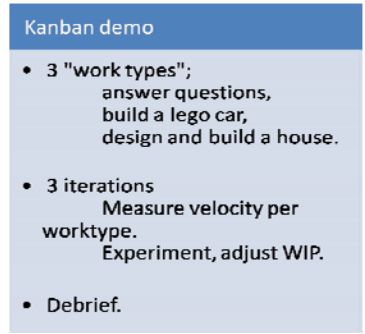
- Should we just get going and deal with the consequences as they appeared?
- Or should we run a workshop first?

It was obvious to us - we should do the workshop first. But how? It was a challenge to get the whole technical operations team to participate in a workshop (who answers if someone calls?). In the end we decided to do a half-day workshop, and keep it simple and exercise-based.

The workshop

One of the benefits of the workshop was that it would help surface our problems early. It also provided a high-trust environment where implications could be discussed directly with team members. I mean let's face it - not everyone was overly enthusiastic about changing the current way of working. But most team members were open to trying. So we executed a workshop demonstrating the most important principles and did a scaled-down Kanban simulation.





At the end of the workshop we did a “fist of five” vote to check if the teams were willing to try this it out for real. No objections were raised at this point so we had approval to move on.

22

Addressing stakeholders

It was likely that the stakeholders would be affected by the Kanban implementation as well. However the changes would be for the better - it meant the team would start saying “no” to work they couldn’t complete, start standing up for quality and removing low-priority items from the team's backlog. Despite this, having a discussion before is always a good idea.

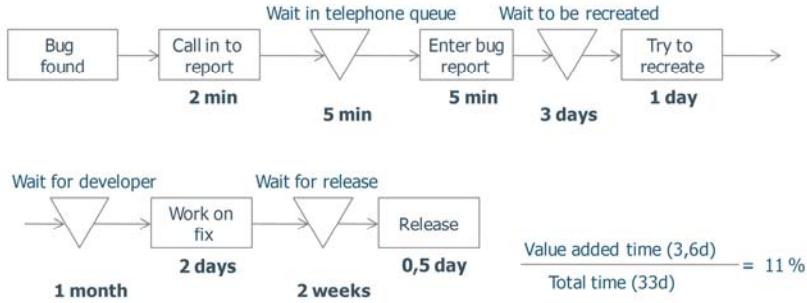
The closest stakeholders were first-line support and department managers. Since they participated in the workshop, they were already positive about moving forward. Same for the development teams (who were more or less expecting improvements anyway). But, for one team, the support team, matters were different. Their most significant problem was that they were overloaded with work. Also, they handled customer issues and the company had a commitment to respond to all issues. Now this was quite likely to change if we implemented Kanban and started enforcing WIP limits.

So, we met with key stakeholders and presented our intentions, expected benefits, and possible consequences. To my relief, our ideas were generally well received, sometimes with a “great if we finally can put these issues to rest” remark.

23

Constructing the first board

A good way to start the construction of a Kanban board is by doing value-stream map. It's basically a visualization of the value chain and provides insight into work states, flow and time through the system (cycle time).



But we started much simpler; a sample Kanban board drawn on paper together with the manager. Reviewed a couple of times and then we got going. Questions brought up at this phase included:

- What types of work do we have?
- Who handles it?
- Should we share responsibility across different work types?
- How do we deal with shared responsibility given specialized skills?

Since the different types of work had different service levels agreements, it felt natural to let each team control the design of their own board. They made out the columns and rows themselves.

The next big decision was whether or not to use shared responsibility across different work types. “Should we let one fixed part of the team deal with the direct questions (reactive work) and let the rest of the team focus on the projects (proactive work)?” We decided at first to try shared responsibility. A key reason was that we had identified that self-organization and continued learning and knowledge transfer on the part of team members were essential to sustain growth. The drawback of this decision was potential disruptions for everybody, but this was the best solution we could think of to start with. A small side note: when we ran the workshop the teams actually self-organized around this problem. They let one person deal with the immediate requests and the rest with the larger issues.

The first Kanban model

Below is the basic model we used for Kanban. Note that the team decided to let items flow upwards (like bubbles in water) rather than the more typical model of flowing from left to right.

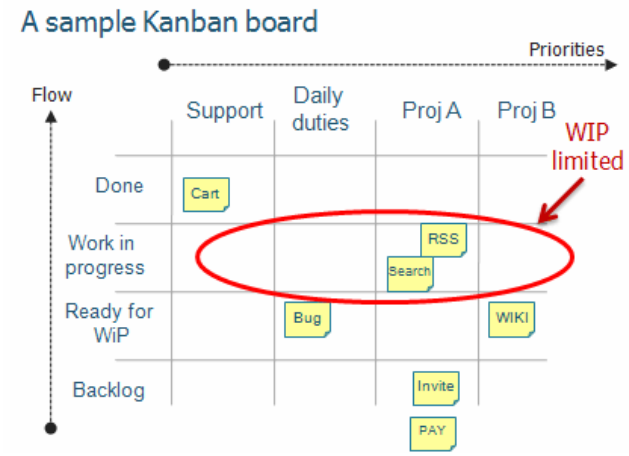


Figure 2. This is the first model of a Kanban board. Priorities run from left to right, flow runs upwards. WIP is counted as the total number of tasks in the work in progress row (circled in black). Model influenced by experiences reported by Linda Cook.

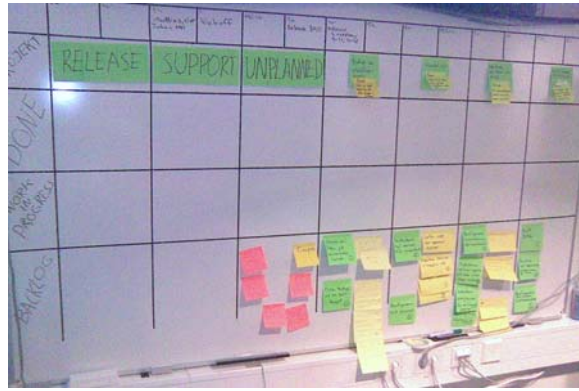


Figure 3. First Kanban board for system administration team.

Rows used

Workflow state (row)	How we defined it
Backlog	Stories the manager decides are needed.
Ready for WIP	Stories which are estimated and broken down to tasks with max size of 8 hours.
Work in progress	The row which had the WIP limit. We started off with a limit of 2 x teamsize - 1 (the -1 is for collaboration). So a 4-person team has a WIP limit of 7.
Done	Runnable by user.

Columns used

Work type	How we defined it
Release	Helping development teams release software.
Support	Smaller requests from other teams.
Unplanned	Unanticipated work that needed to be done but didn't have a clear owner, e.g. minor infrastructure improvements.
Project A	Larger tech ops project, e.g. changing the hardware of a staging environment.
Project B	Another larger project.

Not all of the Kanban boards looked the same. All started out with a simple sketch and evolved along the way.

24

Setting the first Work In Progress (WIP) limit

Our first Work In Progress (WIP) limit was quite generous. We reasoned that by visualizing the flow we would see and experience what went on, and it was unlikely we would be able to guess the best limit from the start. As time passed, we would adjust the WIP limits every time we found good reason to (all we needed to do was point at the board).

The first WIP limit we used was 2n-1. (n= number of team members, -1 to encourage cooperation). Why? Simple, we had no better idea ☺. Also it seemed uncontroversial to start off with. The formula provided a simple and logical explanation to anyone trying to push work onto the team: "... so given that each team member can work on at most two things at the same time, one active and one waiting, why would you expect them to take on *more*?" Looking back, any generous limit would have worked for starters. By monitoring the Kanban board it is easy to figure out the right limits along the way.

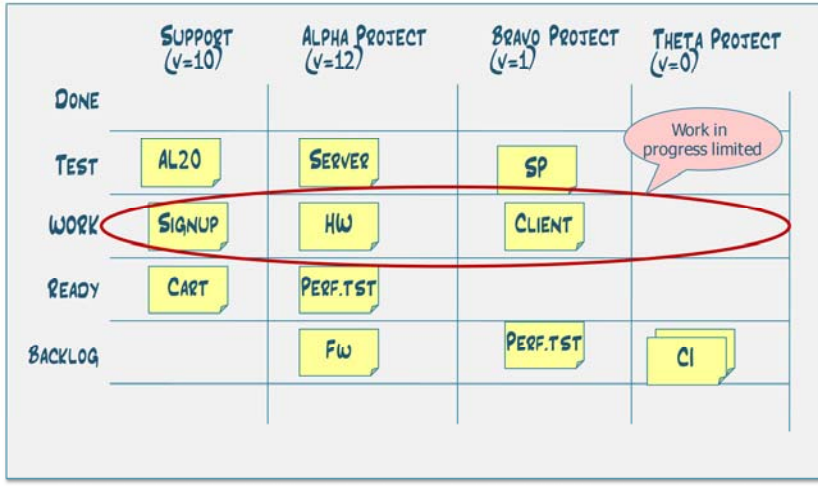


Figure 4. How we applied work in progress limit for DBA and system administration team, one limit across work types.

One observation we made was that it was useless to define the WIP limit in story points. That was just too hard to keep track of. The only limit easy enough to keep track of was simply counting the number of items (= parallel tasks).

For the support team we used WIP defined per column. This because we needed faster reaction if the limit was being overrun.

25

Honoring the Work In Progress (WIP) limit

While respecting a WIP limit sounds easy in theory, it's a tough deal to honor in practice. It means saying "no" at some stage. We tried various approaches to deal with this.

Discuss at the board

If a violation was discovered, we would bring the stakeholder to the board and ask what they wanted to achieve. In the beginning the most frequent reason for violation was just inexperience. In some cases we found different views on prioritizations, a typical case would be a specialist team member working within a specific area. Those were the only times we experienced friction, most of the time the issues were sorted out then and there by discussing in front of the board.

Assigning an overflow section

If saying "no" was too confrontational, and removing items was hard, we moved low priority items to an "overflow" section on the board when WIP limits were exceeded. Two rules applied to overflow tasks:

1. They have not been forgotten; whenever we have time we will deal with them.

2. If we drop them, you will be informed.

After just two weeks it was obvious that the overflow items wouldn't get dealt with ever, so with the support of the team manager these could be finally be removed.

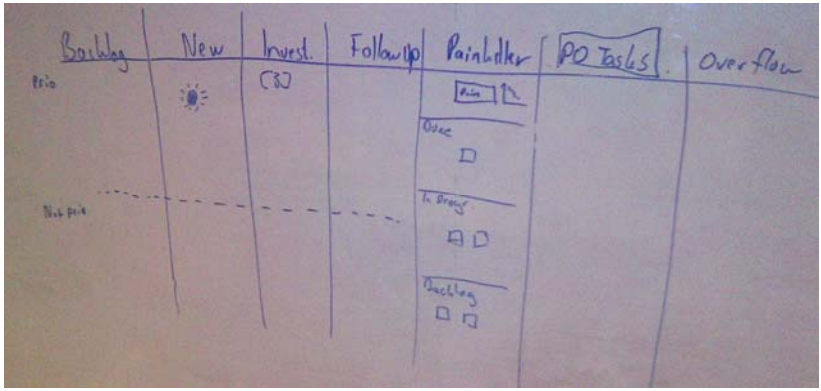


Figure 5. A sketch of the Kanban board for the support team; the overflow section is at the far right.

26

Which tasks get on the board?

We decided early on not to add *all* work done by the team on the board. Monitoring things like a phone call or getting coffee would turn the Kanban into an administrative monster. We were here to *solve* problems, not create problems ☺. So we decided only to put up tasks with size > 1 hour on the board, everything smaller was considered "white noise". The 1h limit actually worked fairly well and was one of the few things that stayed unchanged. (We had to revise the assumption of what impact the background noise had, but more on that later.)

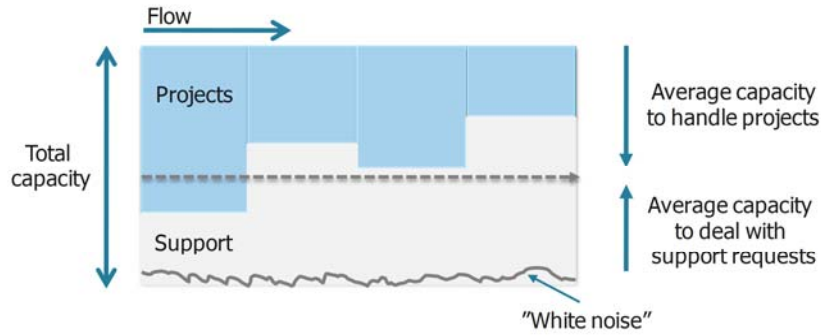


Figure 6. We started by assuming total capacity mainly was consumed by two work types, bigger (projects) and smaller (support). Tracking the velocity on projects could give us an indication of the delivery date if this was required. "White noise" (small support < 1h, meetings, getting coffee, helping colleagues) was always expected to be around.

27

How to estimate?

This is an ongoing topic; and there is certainly more than one answer:

- Estimate regularly.
- Estimate when there is need to.
- Use ideal days/story points for estimates.
- Estimates are uncertain, use T-shirt sizes (Small, Medium, Large).
- Don't estimate, or estimate only when there is a cost of delay that justifies it.

Slightly influenced by Scrum (since that was where we came from, after all) we decided to start with story points. But in practice, teams treated story points equivalent to man-hours (this felt more natural to them). In the beginning, all stories were estimated. Over time, managers learned that if they kept number of concurrent projects low, they didn't keep stakeholders waiting. They also learned that, in case of a sudden change, they could reprioritize and address the problem

The need to estimate a delivery date was no longer a big issue. These led managers to stop asking for up-front estimates. They only did so if they feared they would keep people waiting.

At one point early on, one manager, stressed by a phone call, promised the delivery of a project "by the end of this week". Being a project on the Kanban board, it was easy to estimate progress (count stories completed) and conclude it was about 25% done after one week. Thus an additional three weeks would be required. Confronted with this fact the manager changed the priority of the project, stopped concurrent work, and made the delivery possible. Always check with the board ☺

What does estimated size mean? Lead time or work time?

Our story points reflected work time, i.e. how many hours of uninterrupted effort we expected this story to take; not lead time (or calendar time, or how many hours of wait). By measuring the number of story points reaching "done" every week (velocity) we could deduce the lead time.

We estimated each new story only once, we didn't revise story estimates during execution. This allowed us to minimize the time which the team spent on estimation.

28

So how did we work, really?

Kanban is really very unconstrained, you can work in all sort of ways. You can let team work according to time-based activities, or you can choose to do activities when enough momentum has built up to justify them.

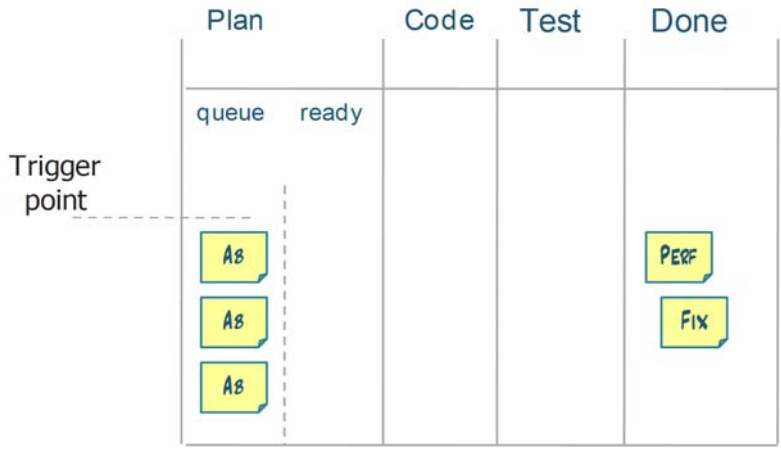
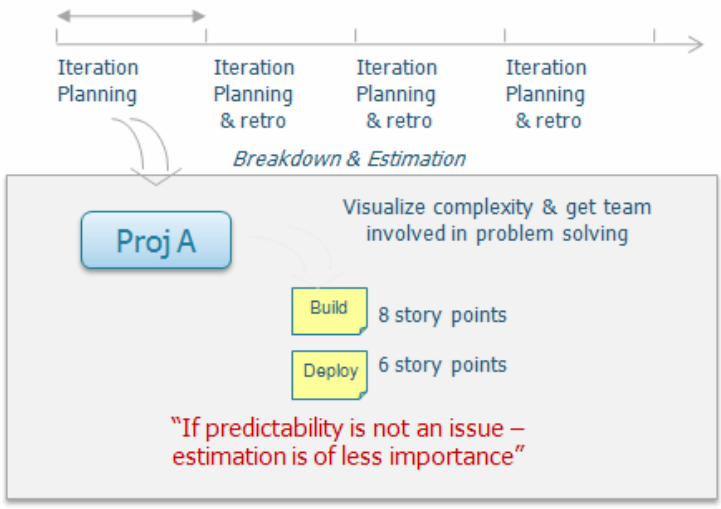


Figure 7 When three tasks have arrived in backlog, this triggers a planning/estimation event.

We chose to schedule two recurring events:

- Daily standup – with the team in front of the board, to surface problems and help create a shared view of other team members tasks.
- Weekly iteration planning, for planning and continuous improvement purposes.



This worked well for us.

Daily standup

The daily standup meeting was similar to a daily scrum. This was run after the daily ”Scrum of Scrums” meeting with participation from all teams (development, test, operations). The Scrum of Scrums gave important input to the Kanban teams, such as which problems should be dealt with first, or which development team was in the most pain at the moment. In the beginning, managers frequently attended these daily standup meetings, proposing solutions and prioritizing decisions. Over time, as the teams grew better at self-organizing, the managers attended less frequently (but were not far away in case of need).

Iteration planning

Once a week, we held an iteration planning. We kept it weekly at a fixed time because we found out that if we did not plan it in, that time was consumed by other priorities☺. We also needed more team talk. A typical agenda was:

- Update charts and board. (Done projects where moved to a “Wall of Done”.)
- Look back at the last week. What happened? Why was it so? What could be done to improve it?
- Readjustment of WIP limit (if needed).

- Task breakdown and estimation of new project [if needed].

Basically the iteration planning was a combined estimation and continuous improvement pulse. Small- to medium-sized issues got resolved on the spot with the support of the first-line managers. But keeping traction on complex, infrastructure-type issues was a harder ordeal. To deal with that, we introduced the ability for teams to assign up to 2 "team impediments" to their managers.

The rules were:

1. Manager can work on two slots at any single point of time.
2. If both are full, you can add a new one as long as you remove the less important one.
3. Team decides when issue is solved.

This was a positive change. Suddenly, teams could see managers were working to help them out even on tough issues. They could point at the impediments and ask "how is it going?" They would not be forgotten or overruled by a new high priority strategy.

One example of a serious impediment was that operations weren't getting the help they needed from developers when operations suspected a bug. They needed a developer to help figure out which part of the system was causing the problem, but since the developers were busy in sprints developing new stuff, problems kept stacking up. Not surprisingly, operations felt that the developers didn't care enough about quality.

When this impediment surfaced, it got escalated first to the line manager and then further to the department manager. He scheduled a meeting together with the head of development. In the discussions which followed, the managers agreed to put quality first. They carved out a round-robin support solution – each sprint, one development team would be "on call" and instantly available to help operations. After securing support from his managers, the head of development handed over a list of contact persons to the support teams. Immediately they put the solution to the test,

suspecting the plan wouldn't work. But it was for real this time; the necessary preparations have been made this time and the impediment was considered resolved. This brought great relief to the operations teams.

29

Finding a planning concept that worked

A story

I remember a turning point for one of the teams. I was sitting with them in their second estimation session. The team was stuck with a project that they didn't know how to estimate. There were too many unknowns and the whole estimation session ground to a halt. Instead of stepping in and taking charge, I asked them to them to refine the process to find a better solution. Led by their manager, they picked up the challenge and started designing their own solution. This event was an important turning point, an important "win" from which they developed into a team with confidence. After this they started to evolve so rapidly that we had to step out of their way.

Two months later, their manager approached me after a retrospective. "I have a problem" he said, pointing at his team's Kanban board. "We have no real problems, what shall we do?"

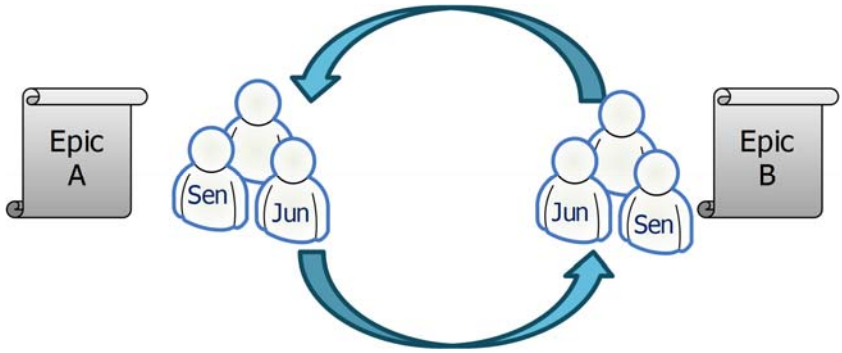
Reinventing planning

Planning poker estimation sessions involving all team members did not work well for any of the operations teams. Some reasons:

1. Knowledge was spread too unevenly within the team.
2. Most often only one person was speaking.
3. Team members wanted to address urgent issues which they had left at their desks.

But by experimentation, the teams independently came up with two different estimation processes. Each worked well for the respective team.

Approach 1 - Shifting and reviewing



- For each project/story, assign a senior + junior pair to estimate it (i.e. one person who knows that particular story well, and one person who doesn't). This helps spread knowledge.
- The remaining team members choose which story they want to help estimate (but with a limit of four people per story to keep the discussions effective).
- Each estimation team does a task breakdown of their story and, if required, estimates it.
- Then the teams switch stories and review each other's work (one person per team "stays behind" to explain his team's work to the reviewers).
- Done!

Typically the whole iteration planning session took around 45 minutes and energy level stayed high throughout the meeting. 1-2 adjustments were typically made when stories were rotated and reviewed by a new set of eyes.

Approach 2 - senior high level pass, then estimation

Two senior team members did a high level review of the story/project before the planning. They would analyze architectural solutions and decide on one for the problem. Once settled, the team would move in and break down the story into tasks using the proposed solution as the starting point.

Figure 8. Task breakdown with peer-review by another team at iteration planning.

30

What to measure?

There are many things that could be measured - cycle time (time from when the need is discovered to when the need is met), velocity, queues, burndowns... The important question is, which metrics can be *used* to improve the process. My advice is to experiment and see what works for you. We learned that burndown charts were overkill for any projects shorter than 4 weeks. The overall progress could be determined simply by looking at the Kanban board (how many stories were in the backlog and how many were done).

Candidate metric	**Pro**	**Con**
Cycle time	Easy to measure. No estimation required. Starts and ends with customer.	Does not take size into consideration.
Total velocity (aggregated across all work types)	A rough, but easy indicator for direction of improvement and variation.	Does not help forecast delivery dates for specific work types.
Velocity per work type	More precise than total velocity.	To be useful, needs to start from a customer need until delivered. Takes a bit longer to track compared to total velocity.
Queue lengths	A fast indicator of demand fluctuation. Easy to visualize.	Does not tell you if the cause is uneven demand or uneven capacity. A zero queue might actually indicate overcapacity.

We started out by measuring "Velocity per work type" and "Queue lengths". Velocity per work type is simple to measure and does the job. Queue lengths are good leading indicators since they can be spotted instantly (once you know where to look for them).

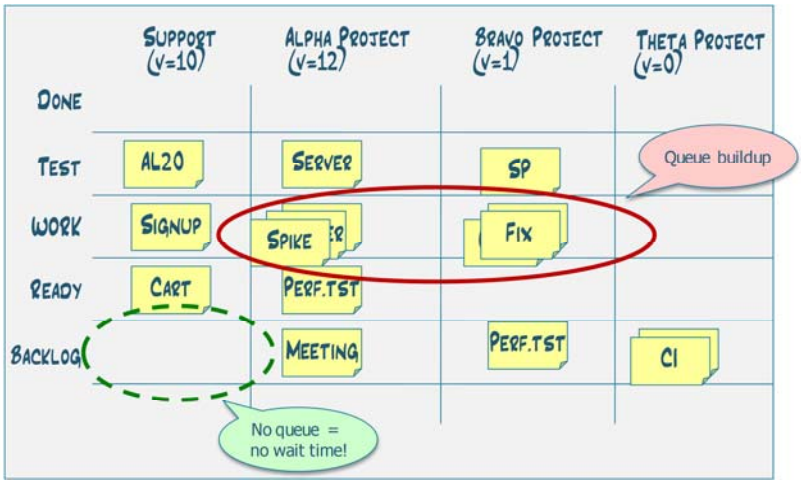
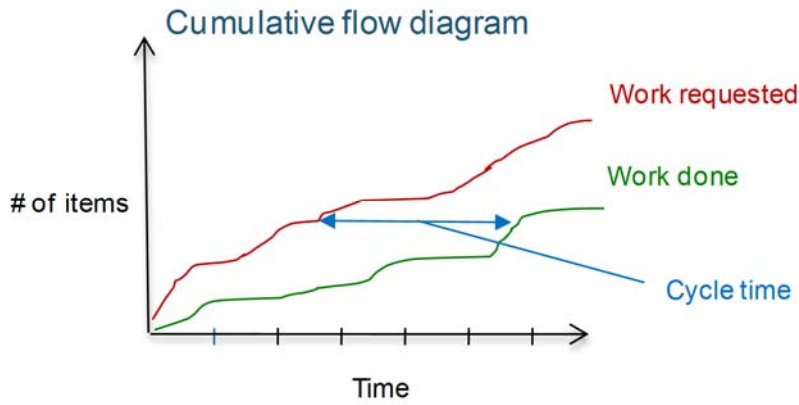


Figure 9. Bottlenecks and opportunities. The black area shows how queues have built up revealing a testing bottleneck. The *absence* of any queue in the support column backlog indicates there is no wait time for new support work. This is a good sign for a high level of customer service.

We didn't use a cumulative flow diagram, but that would have been interesting.



We did not use cumulative flow diagrams since the Kanban board and velocity chart gave us sufficient information, at least in our early phases of maturity. Bottlenecks, unevenness and overwork could still be easily identified and resolving those things kept us busy for the first six months.

31

How things started to change

Three months after introducing Kanban, the system administration team was awarded "best performing team" in the IT department by the management. At the same time, the system administration team was also voted as one of top three "positive experiences" in the company retrospective. The company retrospective is a company-wide event that happens every 6 weeks, and this was the first time that a *team* turned up on the top 3 list! And just 3 months earlier these teams had been bottlenecks that most people were complaining about.

Quality of service had clearly increased. So how did that happen?

The essential moment was when everyone started pulling together. Managers provided a clear focus and protected the team from work that didn't belong there, and the teams took responsibility for quality and deadlines. It took approximately three to four months until this emerged, but after that it flowed smoothly. It's not like every problem in the world was gone (that would put us all out of work, right? ☺) - but we faced new challenges such as "how do we keep a team motivated to improve (when they are no longer the bottleneck)?"

One important piece of the self-organization puzzle was the introduction of the "one operations contact per team" concept. This meant giving each development team their own personal support contact within operations. Kanban made this possible by allowing operations team members to self organize around the work, preventing overwork and enabling continuous improvement. Before, a random person would pull work off the queue, solve it to the best of their ability, and then start on the next. Any miscommunication meant starting all over again with a new support request. When the one-to-one concept was deployed, the support team suddenly had the opportunity to respond quickly when bad input and quality problems threatened the system.

Quickly custom communication protocols evolved; operations staff started to use instant messaging to speak with developers they knew well, email for those who wrote better than they spoke and phone if that was the fastest way to solve the problem ☺.

Before

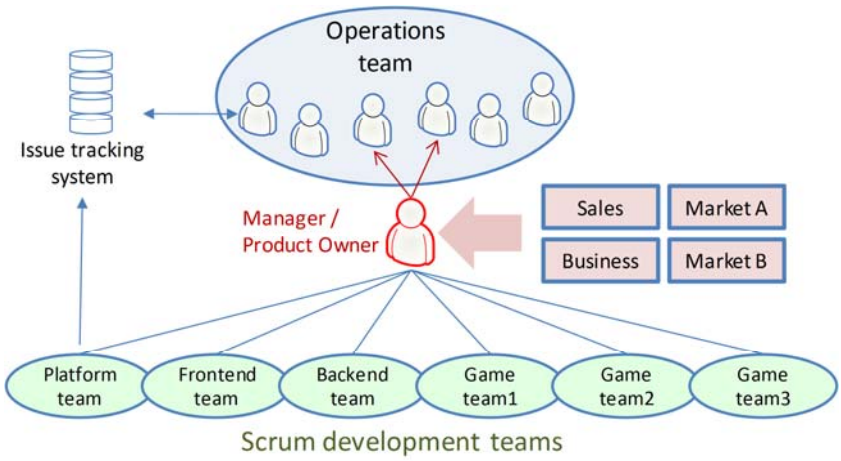


Figure 10. Before: First-line manager is main contact point for team. Anything important that needs to get done passes through him. Smaller issues, typically developer's problems, are received through the issue tracking system. Few direct person-to-person interactions.

After

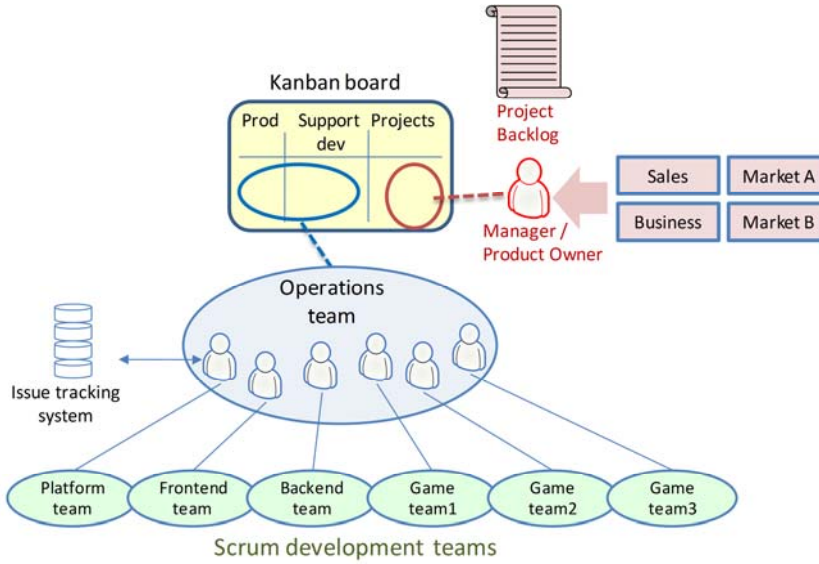


Figure 11. After: "one operations contact per team" deployed. Development teams talk directly to a defined contact person in operations. Many person-to-person interactions. Operations team members self-organize their work using the Kanban board. Manager shifts focus to prioritizing larger projects and providing backup when difficult problems arise.

So what about the impact on team's performance?

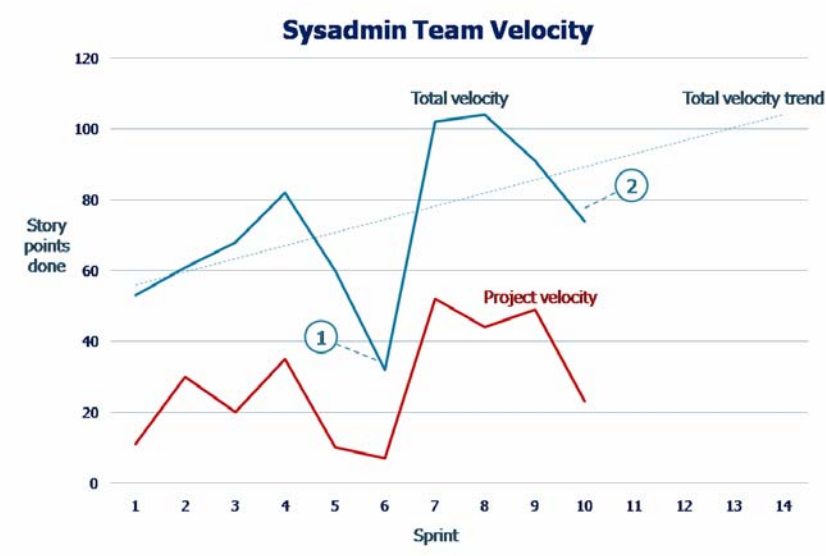


Figure 12. Total velocity and project velocity, measured as story points "done" per week. Total is sum over all columns, Project velocity represents part devoted to "projects" (larger pieces of work, for example upgrading a hardware platform). The two drops correlate to 1) a week where almost all team members were travelling and 2) a major release from development.

Thus, team displayed an overall positive trend. At the same time team invested heavily in knowledge sharing using pair programming.

While we are at it, let's take a look at the DBA team's performance;

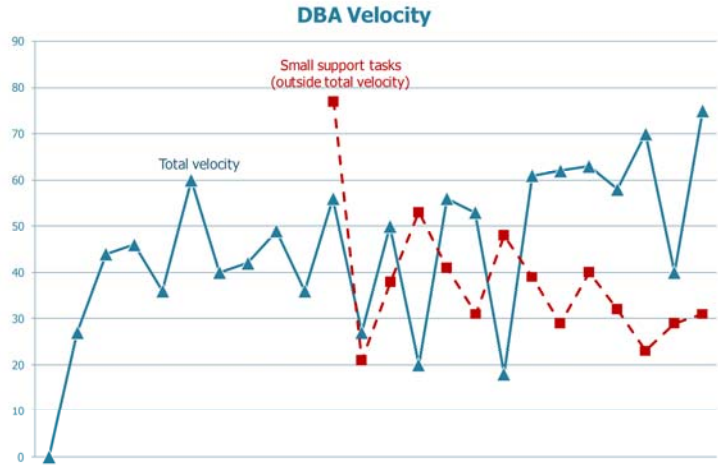


Figure 13. Total velocity and Small support tasks. The drop in the middle corresponds to Christmas.

Total velocity trends upwards although variance is significant. The size of the variance inspired the team to monitor the number of small support tasks (tasks normally too small to make it to the Kanban board). As you can see, the graph indicates a clear inverse correlation between the number of small support tasks and total velocity.

The support team started doing Kanban later than the other two teams so we don't have very much reliable data yet.

Maturity growth

When we started, finding problems areas was easy. But locating the biggest opportunity for improvement was hard. The Kanban board gave us a whole new level of transparency. Not only was it easier to pinpoint problems, but important questions were brought up about flow, variance and queues. We began using queues as a tool to spot problems. Four months after starting to do Kanban, managers were hunting down sources of variance that were hurting their teams.

As teams evolved from individuals to self organizing units, the managers realized they were facing a new set of leadership challenges. They needed to deal more with people issues – handling complaints, defining shared goals, resolving conflicts, and negotiating agreements. Not a painless transition - they openly remarked that learning this took skill and energy. But they took on the challenge and ended up becoming better leaders.

32

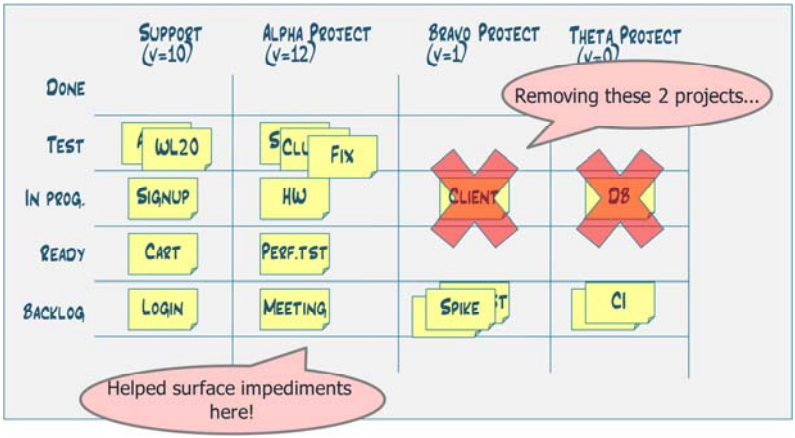
General lessons learned

As work in progress decreases, constraints emerge

All teams started off with quite generous WIP limits. At that time most energy was consumed trying to create flow and making sure the organization was getting the support it needed.

At first the managers wanted to have multiple projects running simultaneously, but within a few weeks it became evident that there wasn't enough capacity to deal with the lower priority projects. It only took one quick glance at the board to see that no work ever got started on the low priority stuff. This prompted the managers to decrease the number of projects per team.

Over time, as flow became steadier for the high priority work, we started tightening the WIP limits. This was done by reducing the number of ongoing projects (columns) from three, to two, then to one. As this happened, constraints outside the team began surfacing. Team members started to report that they weren't getting help from others in time, so managers turned their attention to deal with that.



Some other things which arose included the impact of bad input from other teams upon this team's performance. It was hard to maintain smooth and fast flow when incoming items constantly needed correction.

These problems were not invisible before we started. The issue was rather "which problem should we deal with first" - and reaching common agreement around this. With the Kanban board everybody could see how a specific problem impacted *flow*, which made it easier to gather momentum to deal with the issue across organizational boundaries.

Board will change along the way, don't carve layout in stone

All Kanban boards changed along the way. It usually took two to three redesigns before a team found one that worked well. So investing a lot of time in the first layout is probably wasteful. Make sure you can rearrange the board easily. We used stripes of black tape for the layout. Those were easy to rearrange and could be used on walls as well as whiteboards. Another way I have seen is drawing the board gridlines using thick markers (but make sure they can be erased! ☺)

Below is a typical example of a layout optimization. Priorities shifted frequently in the beginning so, to avoid having to move a whole column of sticky notes back and forth, the team posted a priority number above each column instead.

Figure 14. Early Kanban board with stickers for current priorities

Don’t be afraid to experiment and fail

The lesson I drew from this adventure is that there is really no end point. We fail the moment we perceive there is one. There is only endless experimentation and learning. Never failing means not learning. We failed several times along the road (bad board designs, estimations, redundant burndowns, etc.), but each time we learned something new and important. If we had stopped trying, how could we then be learning?

The success of the Kanban has now inspired the management teams and Scrum development teams to start experimenting with Kanban boards as well. Maybe this book will help!

Final take-away points

Start with retrospectives!

Lots of options and things to think about huh? Hope this book helped clear out some of the fog. At least it worked for us :o)

If you're interested in changing and improving your process, let us make one decision for you right now. If you aren't doing retrospectives on a regular basis, start with that! And make sure they lead to real change. Get an external facilitator in if necessary.

Once you have effective retrospectives in place, you have begun your journey towards evolving just the right process for your context – whether it is based on Scrum, XP, Kanban, a combination of these, or whatever else.

Never stop experimenting!

Kanban or Scrum is not the goal, continuous learning is. One of the great things about software is the short feedback loop, which is the key to learning. So use that feedback loop! Question everything, experiment, fail, learn, then experiment again. Don't worry about getting it right from the beginning, because you won't! Just start somewhere and evolve from there.

The only *real* failure is the failure to learn from failure.

But hey, you can learn from that too

Good luck and enjoy the ride!

/Henrik & Mattias, Stockholm 2009-06-24

H: Is that all we've got?

M: I think so. Let's stop here.

H: Maybe we should tell them who we are?

M: Good point. If we make it seem like we are nice guys we might get consulting gigs.

H: Let's do it then! Then we call it quits.

M: Yeah, we have other work to do, and so do the readers.

H: Actually, my vacation starts just about now :o)

M: Hey, don't rub it in.

About the authors

Henrik Kniberg and Mattias Skarin are consultants at Crisp in Stockholm. They enjoy helping companies succeed with both the technical and human side of software development, and have helped dozens of companies put Lean and Agile principles to work in practice.

Henrik Kniberg

During the past decade Henrik has been CTO of 3 Swedish IT companies and helped many more improve their processes. He is a Certified Scrum Trainer and works regularly with Lean and Agile pioneers such as Jeff Sutherland, Mary Poppendieck, and David Anderson.

Henrik's previous book, "Scrum and XP from the Trenches" has over 150,000 readers and is one of the most popular books on the topic. He has won best speaker awards several times for talks at international conferences.

Henrik grew up in Tokyo and now lives in Stockholm with his wife Sophia and three kids. He is an active musician in his free time, composing music and playing bass and keyboard with local bands.

henrik.kniberg@crisp.se

http://blog.crisp.se/henrikkniberg

http://www.crisp.se/henrik.kniberg

Mattias Skarin

Mattias works as a Lean coach, helping software companies enjoy the benefits of Lean and Agile. He mentors all layers, from developers to management. He has helped a game development company cut game development time from 24 months to 4, restored trust to a whole development department and was one of the early pioneers of Kanban.

As an entrepreneur, he has co-founded and run two companies.

Mattias has a Master of Science degree in Quality Management and has worked as developer for 10 years in mission-critical systems.

He lives in Stockholm and enjoys rock and roll, dancing, racing and skiing.

mattias.skarin@crisp.se

http://blog.crisp.se/mattiasskarin

http://www.crisp.se/mattias.skarin