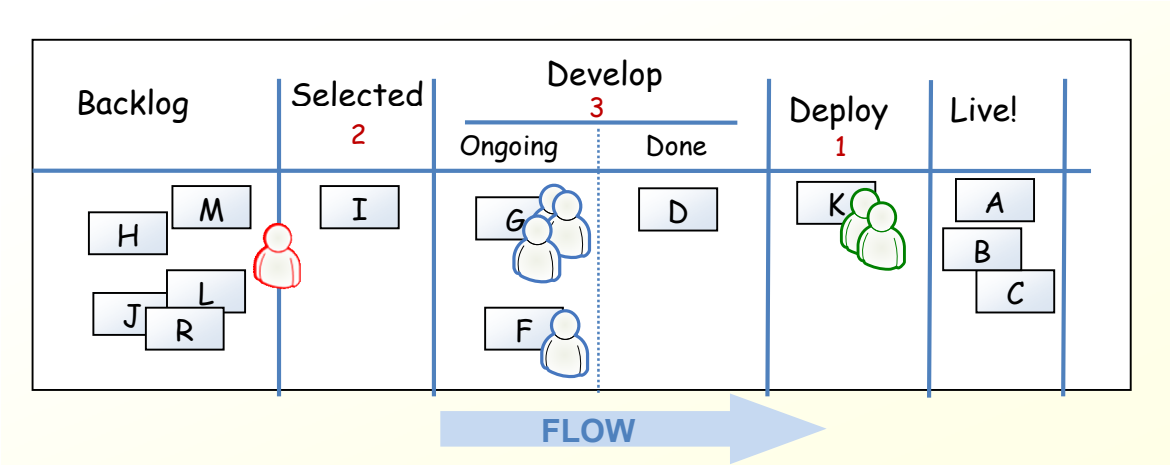
Kanban vs Scrum

How to make the most of both

Henrik Kniberg
henrik.kniberg@crisp.se

Version 1.1 (2009-06-29)
Original version: 2009-04-18



1. Introduction

There's a lot of buzz on Kanban right now in the agile software development community. Since Scrum has become quite mainstream now, a common question is "so what is Kanban, and how does it compare to Scrum?" Where do they complement each other? Are there any potential conflicts? Here's an attempt to clear up some of the fog.

- **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 & maintainance 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 article!"
- **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 article

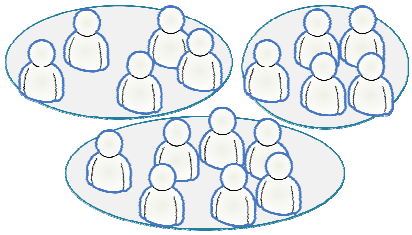
If you're interested in agile software development you've probably heard about Scrum, and you may also have heard about Kanban. I hope this article will clarify both tools by comparing them to each other, so you can figure out how these might come to use in your environment.

2. So what is Scrum and Kanban anyway?

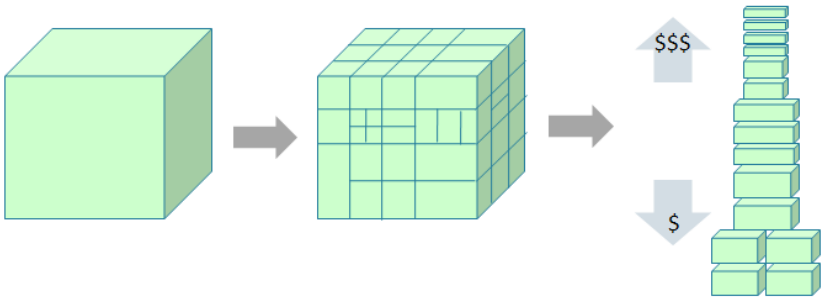
OK let's try to summarize Scrum and Kanban in less than 100 words each.

Scrum in a nutshell

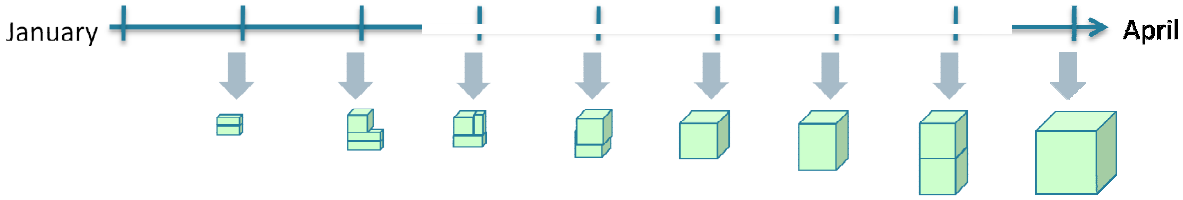
- **Split your organization** into small, cross-functional, self-organizing teams.



- **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.

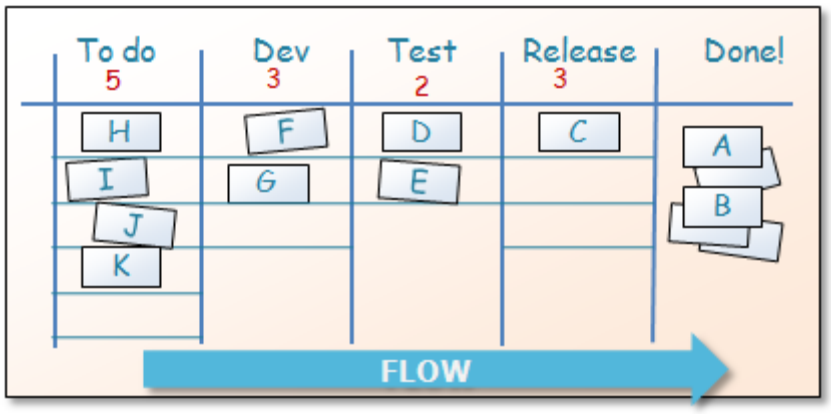
121 words... close enough.

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 WIP** (work in progress) – 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

3. So how do they relate to each other?

Scrum and Kanban are both process tools

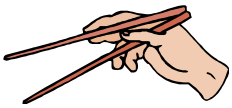
Tool = anything used as a means of accomplishing a task or purpose. *Process* = how you work.

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 programming a computer. A toothbrush is a also a tool, it helps you reach your teeth so you could clean them.

Compare tools for understanding, not judgement

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 fork while others prefer chopsticks.



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

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

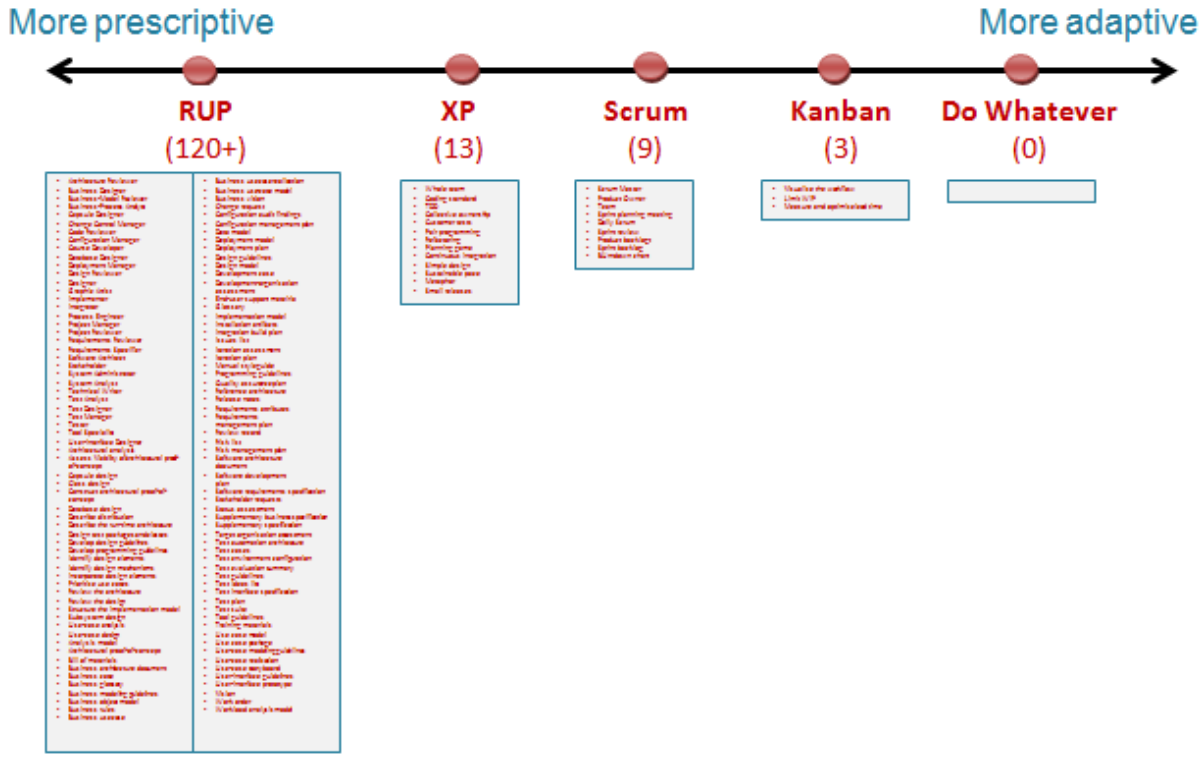
Scrum is more prescriptive than Kanban

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 less options are 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 for 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 contraints 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.

Musashi said it nicely (famous 17th century Samurai, famous for his twin-sword fighting technique)

> Do not develop an attachement 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)

4. 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 multiple teams & product owners synchronize with each other. In a small project that role might be waste, or worse, might lead to suboptimization 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 article 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.

5. 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*.

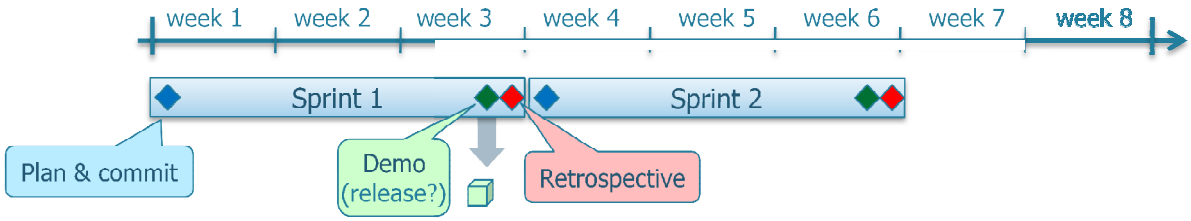
- **Beginning of iteration:** An iteration plan is created, i.e. team pulls out specific number items from the product backlog, based on the product owner's priorities and how much the team thinks they can complete in one iteration.
- **During iteration:** Team focuses on completing the items they committed 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") .

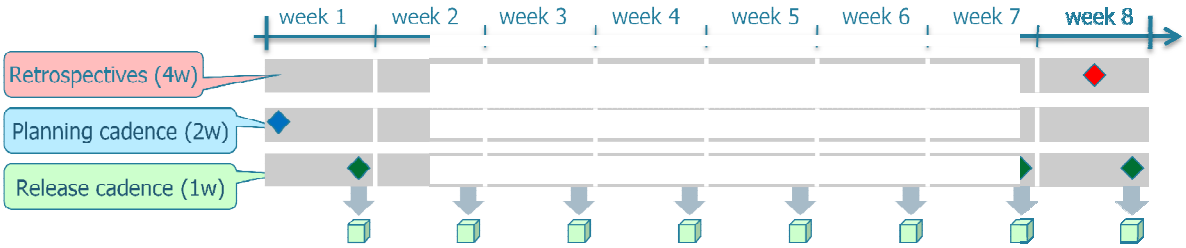
Team #1 (single cadence)

"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"



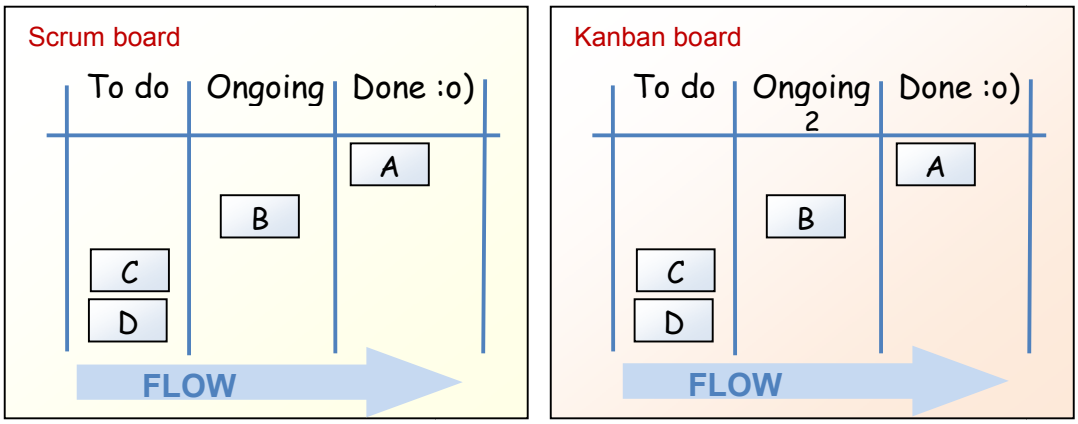
Team #3 (mostly event-driven)

"We trigger a planning meeting whenever we start running out of stuff to do. We trigger a release whenever there is a MMF (minimum marketable feature set) 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."

6. 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.

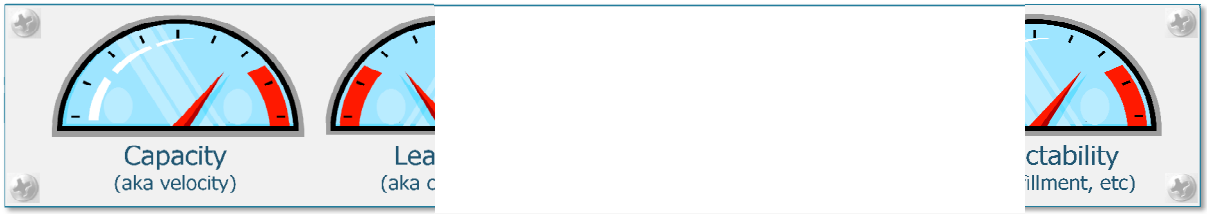
So in Scrum *WIP is limited per unit of time.*

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.

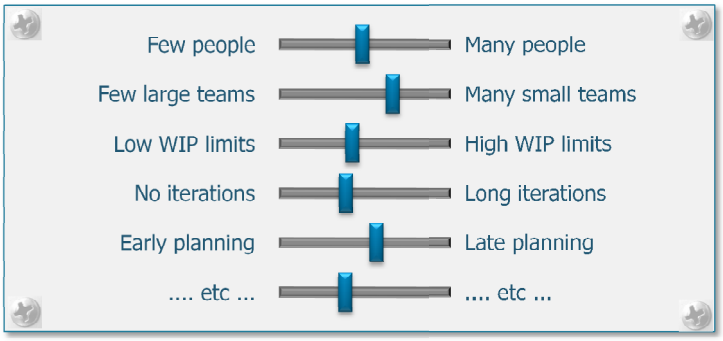
7. Both are empirical



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 knows 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.

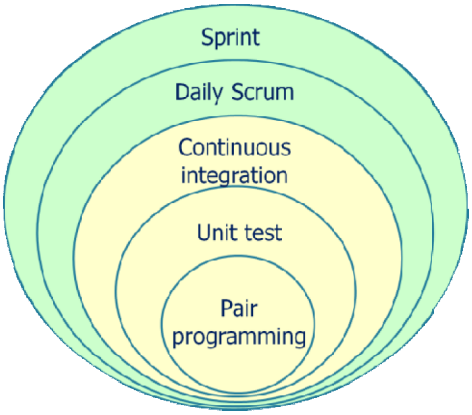
As I mentioned earlier, Kanban imposes fewer constraints than Scrum. This means you get more parameters to think about, more knobs to turn. That can be both a disadvantage or an advantage depending on your context. When you open up the configuration dialog of a software tool, do you prefer having 3 options 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 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 a "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 empy. Look for "air bubbles" on your board.

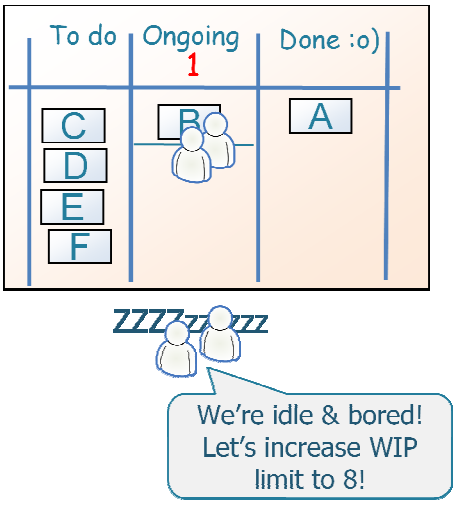
The nice thing about real-time metrics is that you can choose the length of your feedback loop, based on how often you are willing to analyze the metrics and make changes. Too long feedback loop means your process improvement will be slow. Too short feedback loop means your process might not have time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

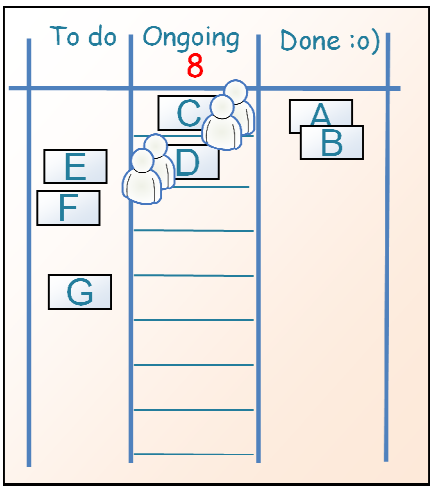
Let's say we have a 4 person team, and we decide to start with a WIP limit of 1.



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.

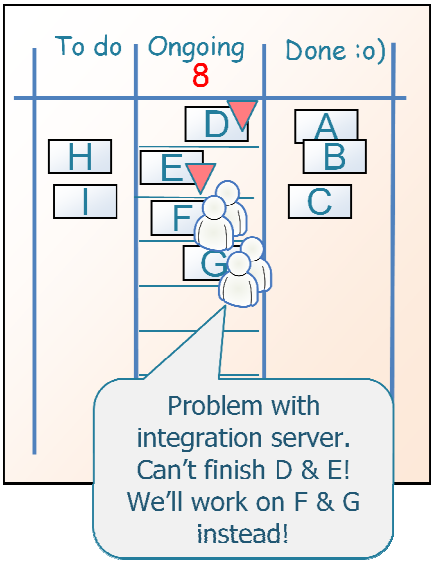
Great! But then it turns out that it's usually not feasible for all 4 people to work on the same item (in this sample context), so we have people sitting idle. If that only happens once in a while that's no problem, but if it happens regularly, the consequence is that the average lead time will increase. 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?

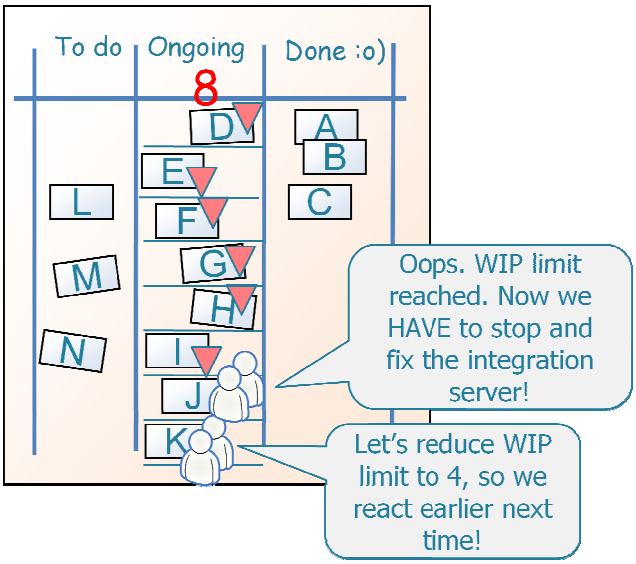


That works better for a while. We discover that, on average, working in pairs gets the work done most quickly. So with a 4 person team, we usually have 2 ongoing items at any given time. The WIP of 8 is just an upper limit, so having fewer items in progress is fine!

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.

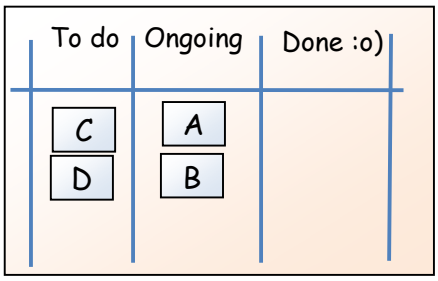
After a while we might find that items pile up in "To do". Maybe it's time to add a WIP limit there as well then.

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!

8. Scrum resists change within an iteration

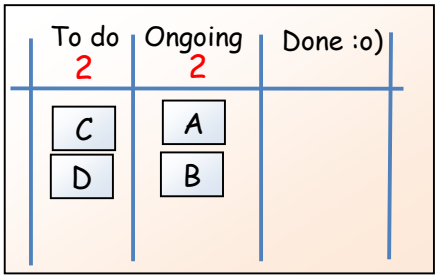
Let's say our Scrum board looks like this:



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).

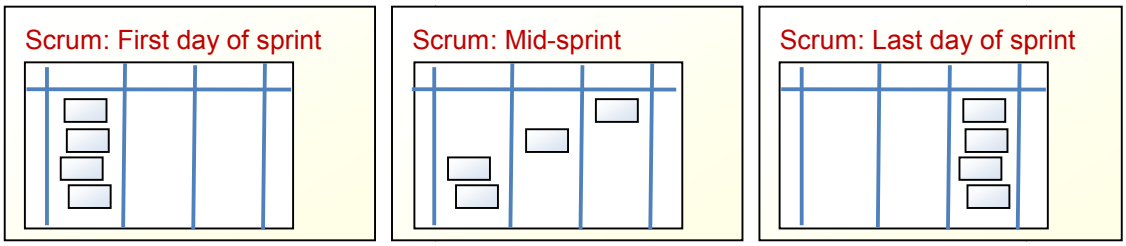
In Scrum, the response time is half the sprint length on average.

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 fix-commitment iterations, just like Scrum.

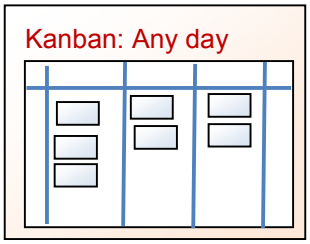
9. Scrum board is reset between each iteration

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



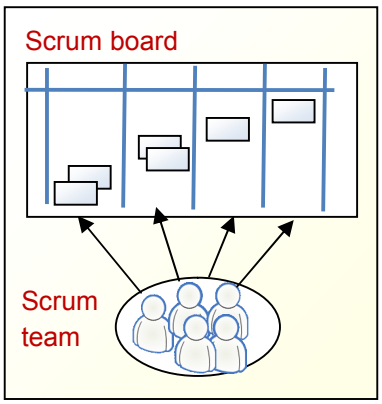
When the sprint is over, the board is cleared – all items are removed. A new sprint is started and after the sprint planning meeting we have a new Scrum board, with new items in the left-most column. Technically this is waste, but for experienced Scrum teams this usually doesn't take too long, and the 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.



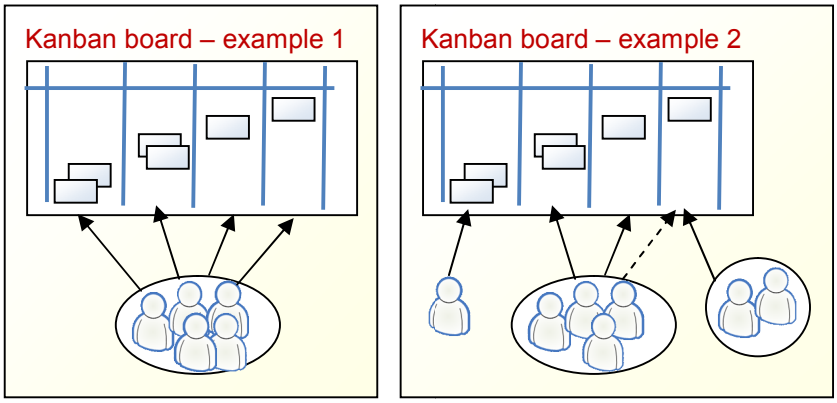
10. Scrum prescribes cross-functional teams

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 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.

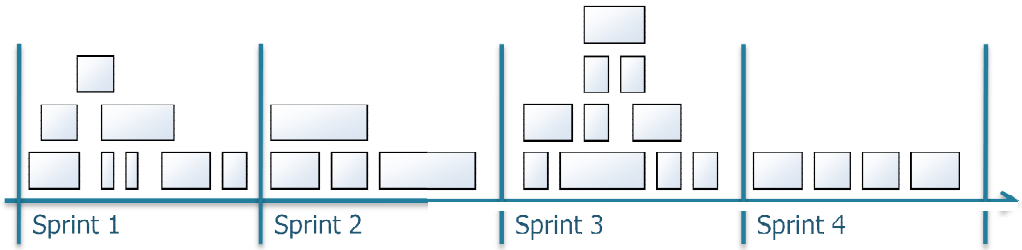
Example 2: the product owner sets priorities in column 1. A cross-functional development team does development (column 2) and test (column 3). Release (column 4) is done by a specialist team. There is slight overlap in competencies, so if the release team become a bottleneck one of the developers will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

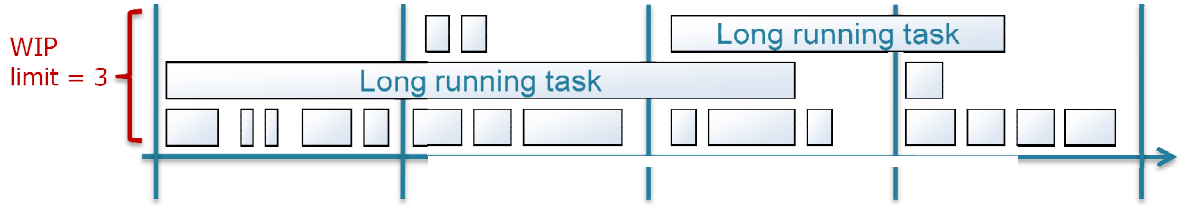
11. Scrum backlog items must fit in a sprint

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).

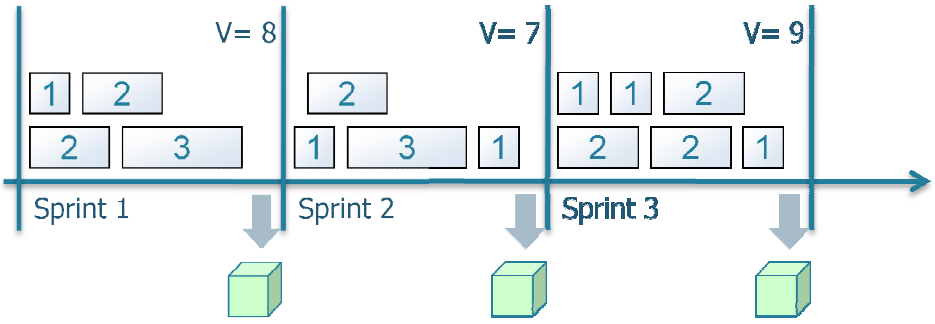


Kanban teams try to minimize lead time and level the flow, so that indirectly creates an incentive to break items into relatively small pieces. But there is no explicit rule stating that items must be small enough to fit into a specific time box. On the same board we might have one item that takes 1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



12. 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 if 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.

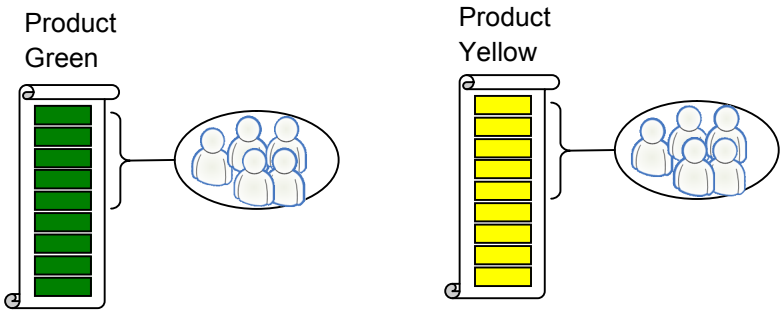
In Kanban, estimation is not prescribed. So if you need to make commitments you need to decide how to provide predictability.

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 (minimum marketable features) and measure the average lead time per MMF, and use that to establish SLA (service-level agreements) – 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.

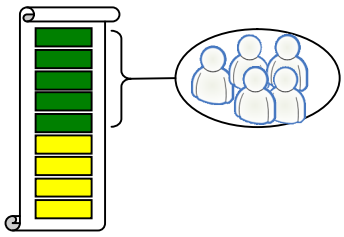
13. Both allow working on multiple products simultaneously

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:

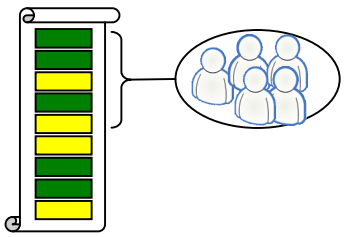


What if you only have one team then? Well, think of Product Backlog more like a Team Backlog. It lists the priorities for upcoming iterations for one particular team (or set of teams). So if that team is maintaining multiple products, merge both products into one list. That forces us to prioritize between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

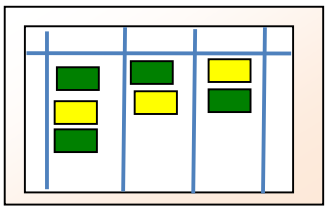
One strategy would be to have the team focus on one product per sprint:



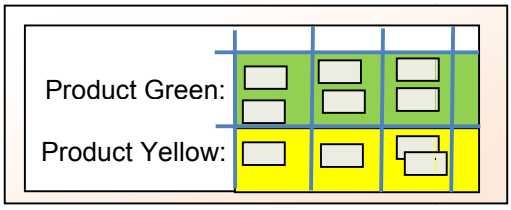
Another strategy would be to have the team work on features from both products each sprint:



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



... or by having "swimlanes" :



14. 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 correponds 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 non-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)

15. Minor differences

Here are some differences that seem to be less relevant compared to the other ones mentioned above. 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 maintainance 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

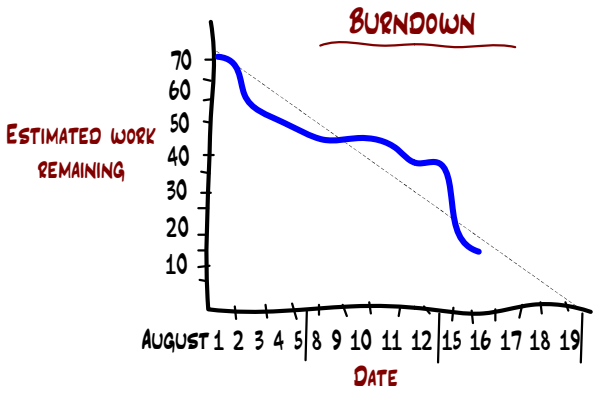
A Scrum team has a short meeting (at most 15 minutes) every day at the same time & same place. The purpose of the meeting is to spread information about what is going on, plan the current day's work, and identify any significant problems. This is sometimes called a daily standup, since it is usually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

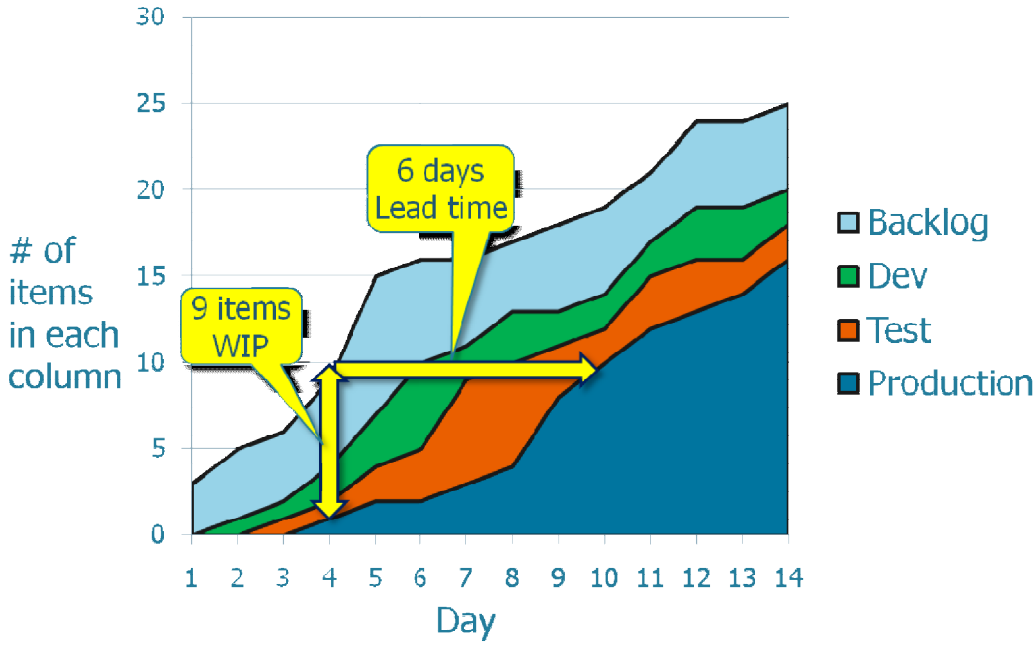
In Scrum, sprint burndown charts are used as one of the primary tools for tracking the progress of an iteration.

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 they are of course allowed to use any type of chart they like (including burndowns).

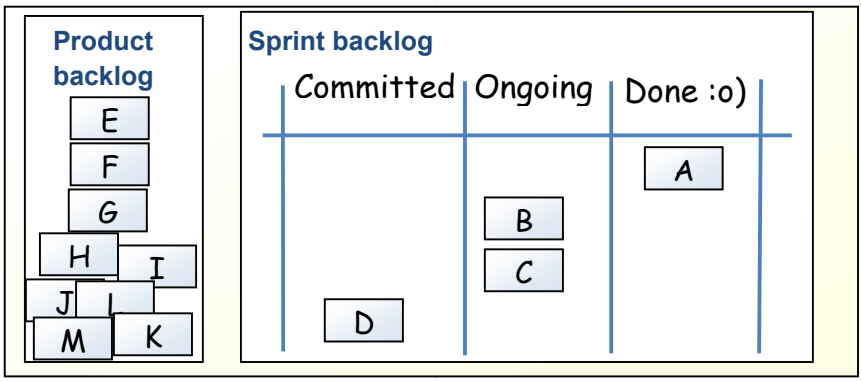
Here's an example of a Cumulative Flow diagram. This type of chart illustrate nicely how smooth your flow is and how WIP affects your lead time. Just add up the number of items in each column of your taskboard and update the chart.



16. Scrum board vs Kanban board – a less trivial example

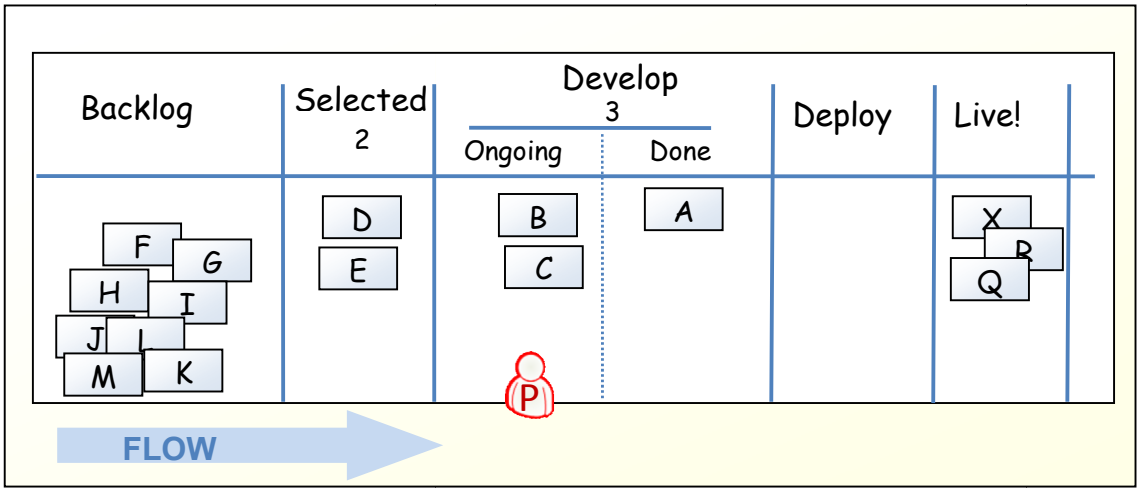
In Scrum, the sprint backlog is just one part of the picture – the part that shows what the team is doing during the current sprint. The other part is the product backlog – the list of stuff that the product owner wants to have done in future sprints.

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. affect 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 PO 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:



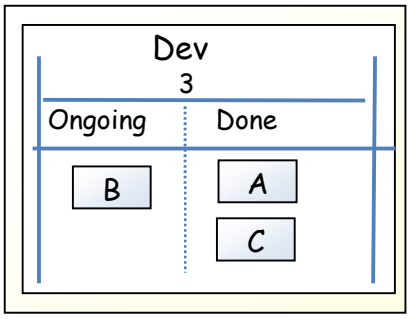
Now the whole workflow is on the same board – we're not just looking at what one Scrum team is doing in one iteration.

In the example above the "Backlog" column is just a general wishlist, 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":

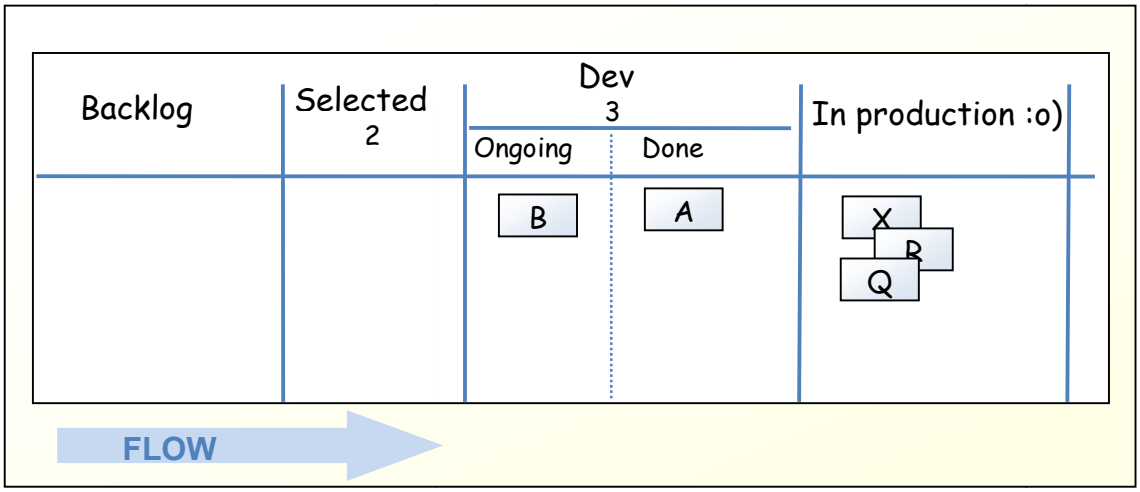


That means there can only be 1 item in "Ongoing". That means there will be excess capacity, developers who *could* start a new item, but aren't allowed to because of the Kanban limit. That gives them a strong incentive to focus their efforts and helping to get stuff into production, to clear the

"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:

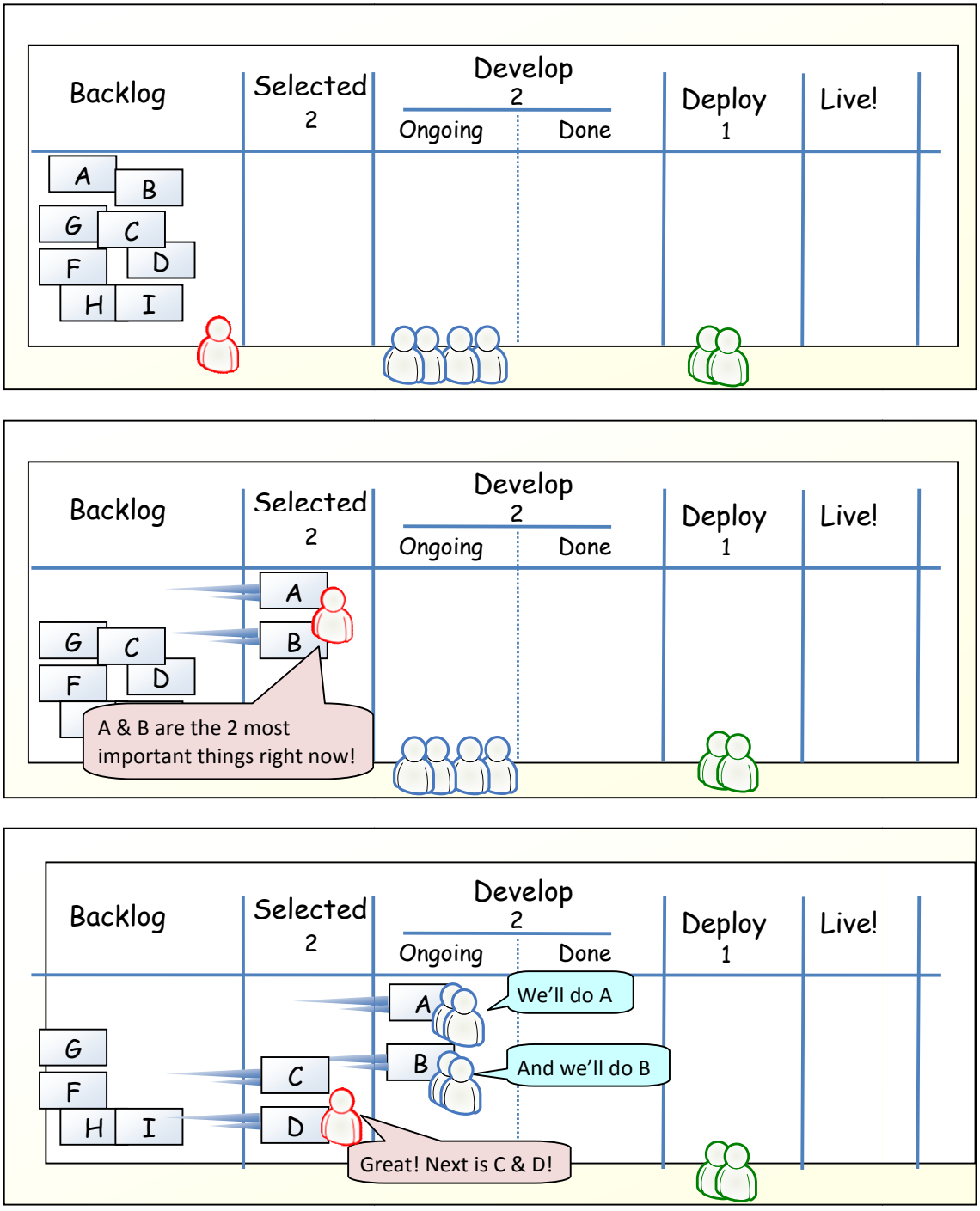


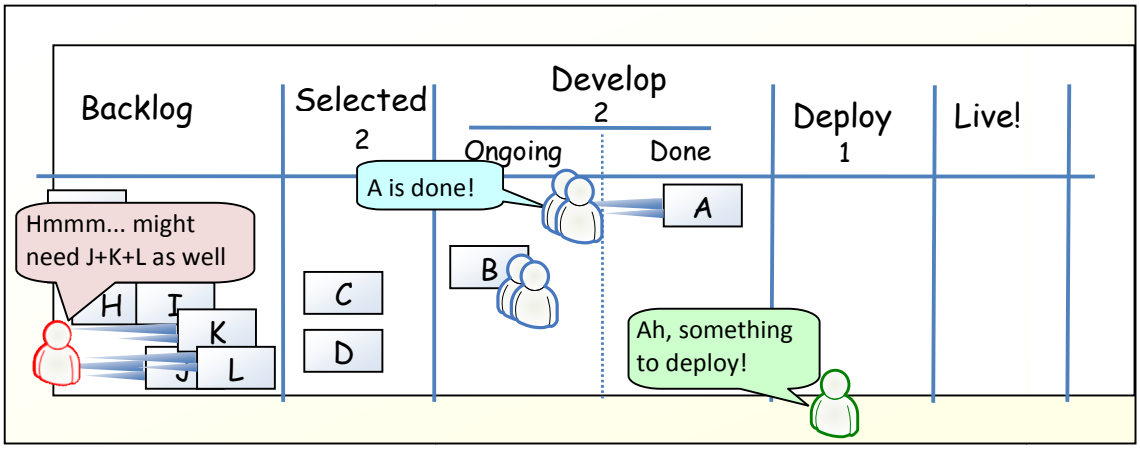
B is being developed at the moment, A is being put in production at the moment. Whenever the team is ready for the next item they ask the product owner that is most important, and get an instance response. If this ideal scenario persists we can get rid of the two queues "Backlog" and "Selected" and get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

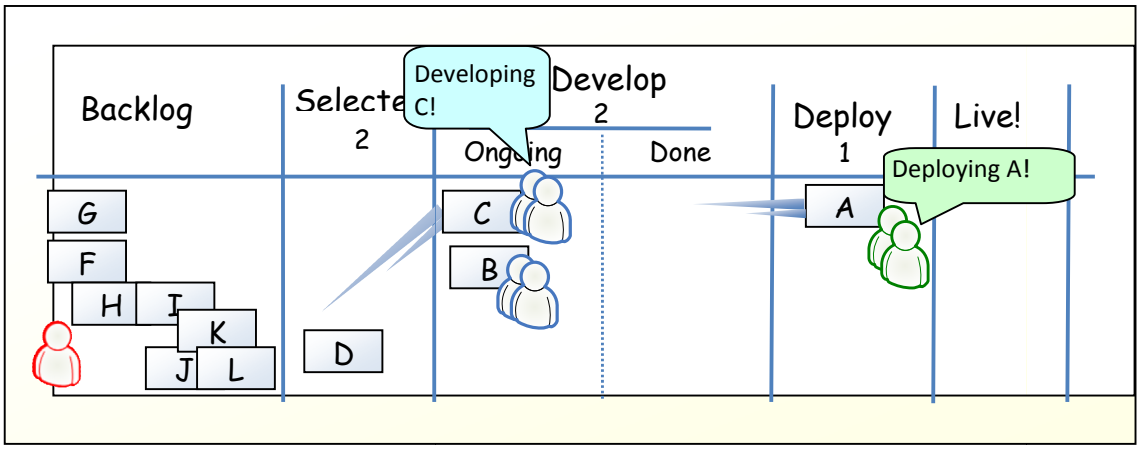
Cory Ladas puts it nicely: "The ideal work planning process should always provide the development team with best thing to work on next, no more and no less".

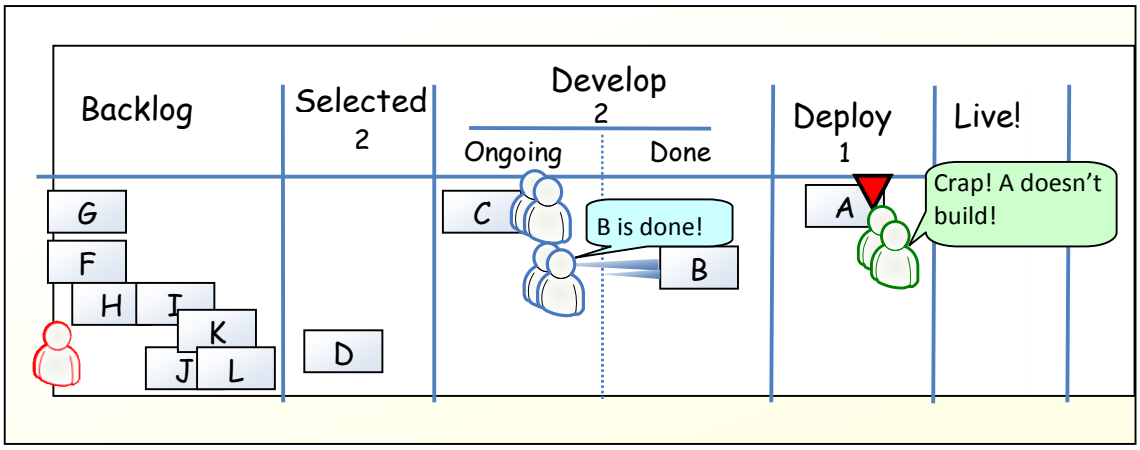
The WIP limits are there to stop problems from getting out of hand, so if things are flowing smoothly the WIP limits aren't really used.

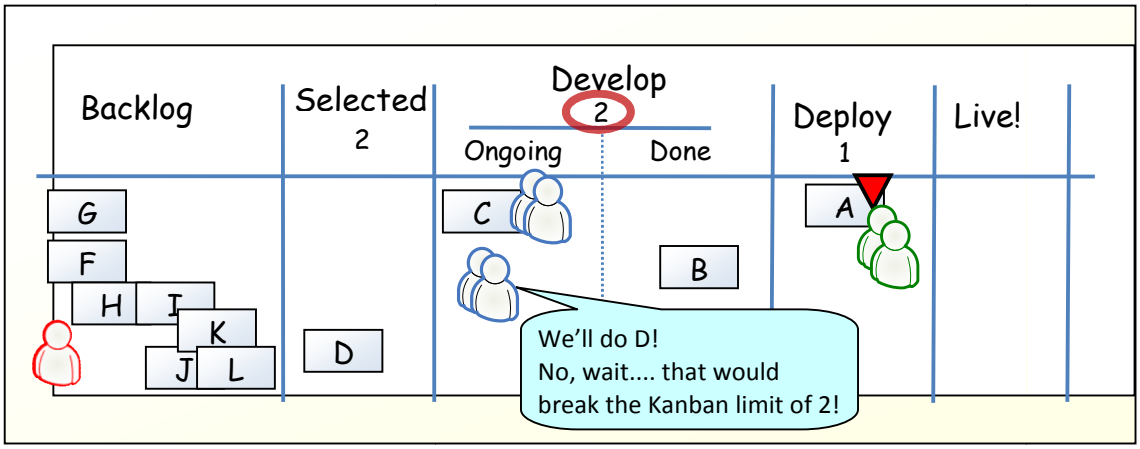
One day in Kanban-land

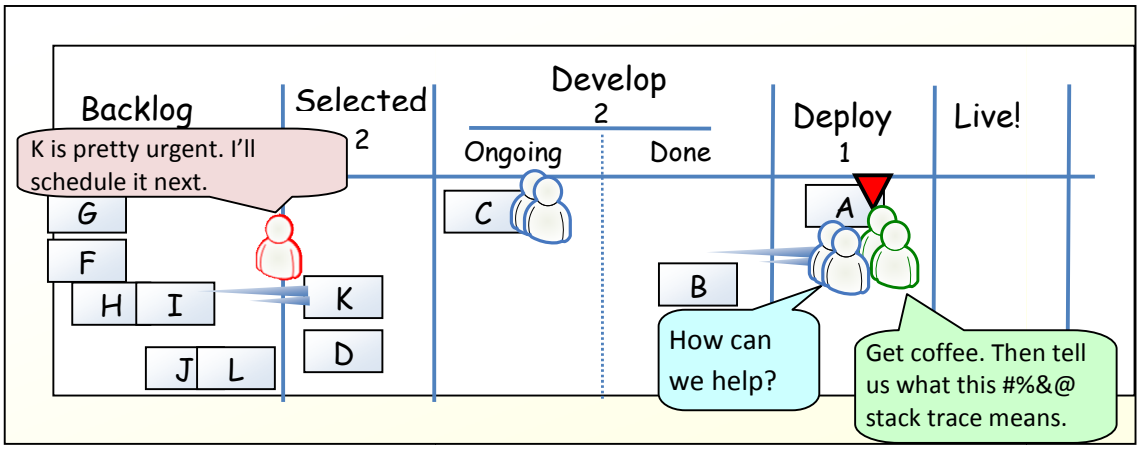


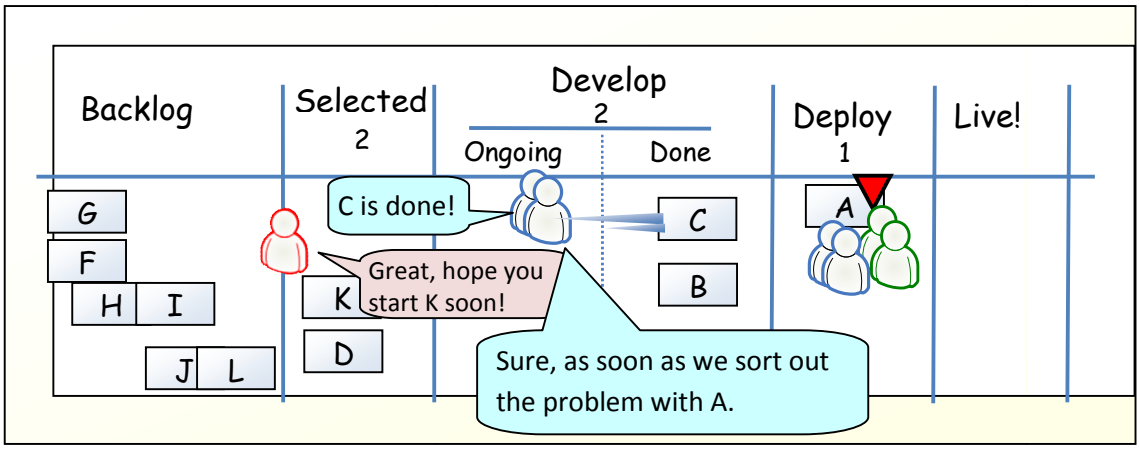

Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
A is done!
A
Hmmm... might need J+K+L as well
H
I
K
J
L
C
D
B
Ah, something to deploy!

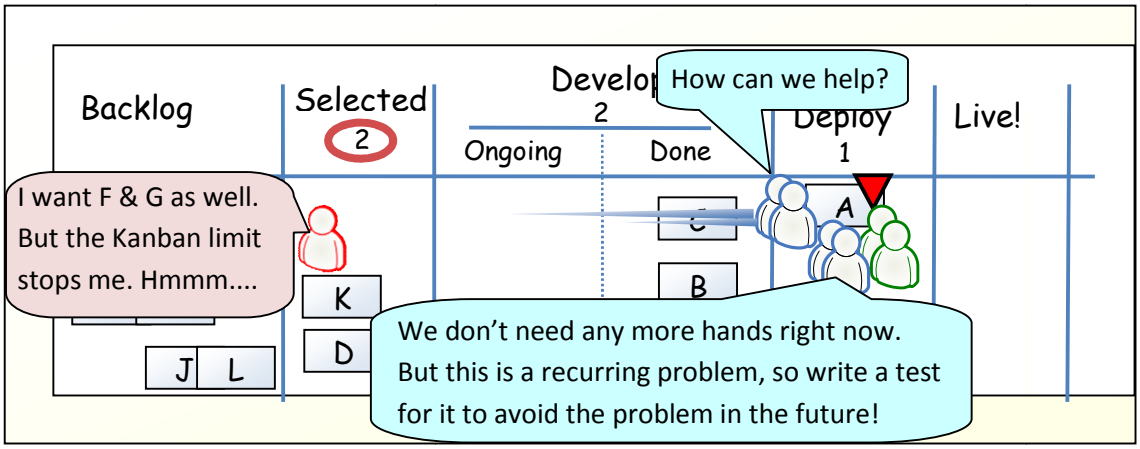

Backlog
Selecte
2
Developing C!
Develop
2
Ongoing
Done
Deploy
1
Live!
Deploying A!
A
G
F
H
I
K
J
L
D
C
B

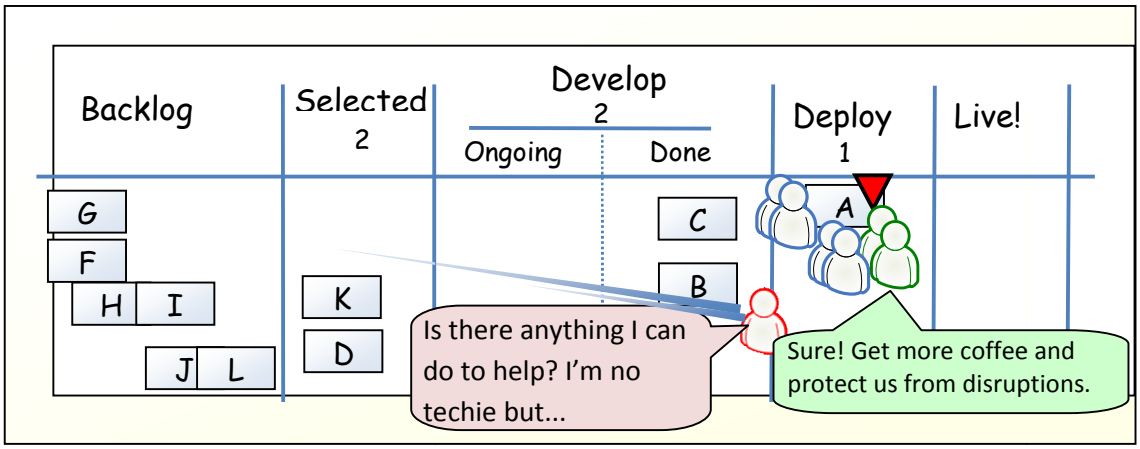

Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
Crap! A doesn't build!
A
B is done!
B
G
F
H
I
K
J
L
D
C

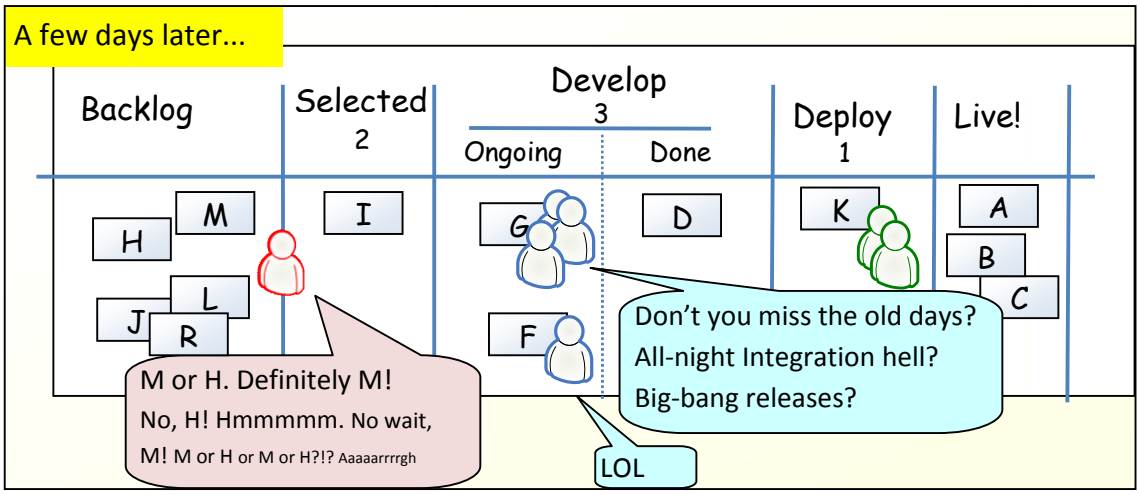

Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
G
F
H
I
K
J
L
D
C
B
A
We'll do D!
No, wait.... that would
break the Kanban limit of 2!


Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
K is pretty urgent. I'll
schedule it next.
G
F
H
I
K
J
L
D
C
B
A
How can
we help?
Get coffee. Then tell
us what this #%&@
stack trace means.


Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
C is done!
Great, hope you
start K soon!
G
F
H
I
K
J
L
D
C
B
A
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
C
B
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
L
J
R
I
G
F
D
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?

17. 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, 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**.

18. Take-away points

Start with retrospectives!

Lots of options and things to think about huh? Hope this article 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 to...

Good luck and enjoy the ride!

/Henrik 2009-04-03
henrik.kniberg <at> crisp.se

http://blog.crisp.se/henrikkniberg - more articles & random ramblings
http://www.crisp.se/henrik.kniberg - my home page