
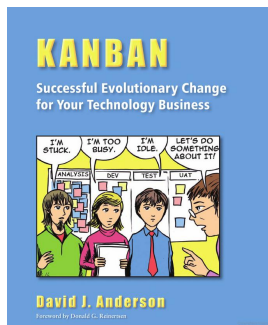


Kanban

 @rchatley #doc302

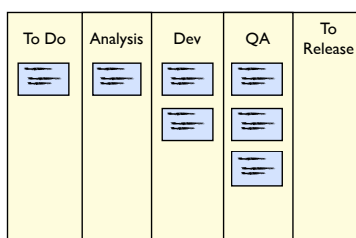
Here we give a brief introduction to Kanban. Kanban is a method that has relatively recently seen adoption in the software industry, but has its roots in manufacturing. A lot of the ideas were developed originally in places like Toyota in Japan, producing cars. Toyota were concentrating on trying to produce cars efficiently on the production lines in their factories, minimising waste.



The original kanban systems at Toyota were designed by Taiichi Ohno (pictured bottom right observing the production line). A lot of the work in popularising it amongst the software development community was led by David Anderson, who wrote the pictured book. He also has an earlier book “Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results” which focussed more on some of the work he did at Corbis, and his thinking at that stage is presented using more mathematics, which aren’t always useful to individual teams (they might be to higher levels of management).

Another strong influence on Kanban is Goldratt’s theory of constraints (TOC) - https://en.wikipedia.org/wiki/Theory_of_constraints

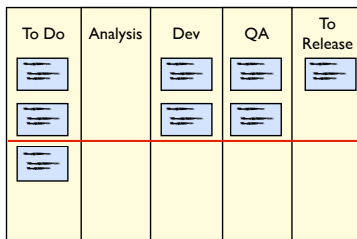
Kanban/TOC



Kanban is not a method for managing people, it is for managing work. We do not focus on what people are doing, we focus on the work - what state it is in, and what does it need next? We use a board to visualise the workflow. This kanban board shows that a number of stories have been developed, but they are backing up in QA - none have been released. The team should focus on getting these features through QA and into production before doing more dev or analysis work.

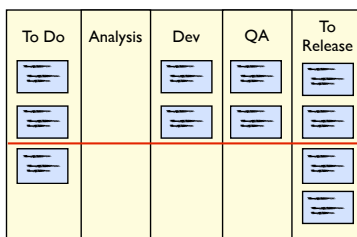
But, it is a natural trait of developers to want to keep busy (and of managers to want to keep them busy), so we might be tempted to start more of the things in the To Do column. This causes us to have more work in process. More things have been started, but not necessarily finished. If we came to the end of a Scrum sprint in the situation shown here, we would have nothing to release. Kanban focusses on getting work to flow through the system.

Limit WIP



One way to help highlight these problems is to enforce a work-in-process (WIP) limit on each activity. We impose a policy where we can only pull a task into the next phase when there is a space to do so. When the team comes to look at this board and decide what to do next, they start on the right hand side - not the left. Can we move anything from QA into To Release? If not, what needs to be done before that can happen? What can we best do as a team to get something ready to release - get it to the end of the pipeline. Once there is a space in the QA column, we can ask the same questions about the tasks in the Dev column. These constraints focus us on moving things through the pipeline.

Release as a Flow



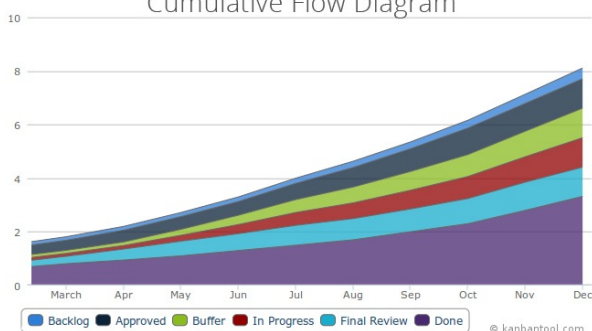
Unreleased features are inventory

More features per release implies more risk

Although this board looks good, there is still a problem. We have lots of things that are “ready to release”, but they are not released. This suggests that perhaps our release process is troublesome or time consuming. If we work on making this easy, then release becomes routine, and we can get features released as soon as they are finished. Keeping the work in progress low and releasing often, a few features at a time helps to keep customers happy as they see a continuous flow of delivery. It also means that we have less half-developed features to maintain, and smaller changes in each release.

Instead of working in fixed timeboxes, whenever there is space to pull something from the To Do column, we start on the next most important thing. Instead of planning what we can do in a fixed time box, we measure how long it takes for each task to pass through the pipeline. Then we can say “on average it takes us X days to deliver a new feature after it reaches the top of the To Do list”.

Cumulative Flow Diagram



<http://kanbantool.com/kanban-library/analytics-and-metrics/explaining-cumulative-flow-diagrams>

With Kanban we try to analyse our workflow and make it more efficient. Our aim is that each piece of work should spent the minimum possible time in each phase - particularly we want to eliminate waiting. One way we can investigate this is using a cumulative flow diagram. Each day we take a snapshot of the kanban board and note the number of items in each phase. Ideally over time the number of things in “done” should increase, but if we keep up our throughput and flow, then only a small number of tasks should be in any of the other phases at any time.

If one of the phases starts accumulating work, we can use this as a sign that we need to focus our efforts on making our work in this phase more effective. For example we might want to spend more time building some tools to speed up testing and deployment, rather than starting new features.