Kanban – what is it and why should I care?

Landon Reese
Kathy Iberle

Abstract

Kanban is gaining popularity in the software development world. It deserves to be considered as a means to manage software development. Kanban is a lightweight agile model which provides visibility to work in process, the capacity of a given resource pool, and the current workflow. The Core Test Strategy Lab at Hewlett Packard has adopted Kanban, and has seen concrete evidence of its efficacy. Using our experience in the Core Test Strategy Lab, this paper will do the following:

- Explain how Kanban can be used as an agile software development method
- Provide some guidelines for running Kanban effectively
- Lay out some situations where the use of Kanban will be of benefit
- Elaborate on how the low upfront cost of Kanban accelerated its adoption
- Identify some situations in which Kanban *will not* be of benefit

Biography

Landon Reese is currently a project manager at Hewlett-Packard in the Core Test Strategy Lab. Within his current role he has implemented a Kanban process to manage tool development, and the build of a data center at the Boise site. Prior to his current role, Landon spent two years as a firmware engineer on enterprise class laser printers, responsible for scan ASIC turn-on and testing of the scanner interface.

Landon has a B.S. in Electrical Engineering from Santa Clara University.

Kathy Iberle is a senior software quality engineer at Hewlett-Packard, currently working at the HP site in Boise, Idaho. Over the past twenty-five years, she has been involved in software development and testing for products ranging from medical test result management systems to printer drivers to Internet applications. Kathy has worked extensively on training new test engineers, researching appropriate development and test methodologies for different situations, and developing processes for effective and efficient requirements management and software testing.

Kathy has an M.S. in Computer Science from the University of Washington and an excessive collection of degrees in Chemistry from the University of Washington and the University of Michigan.

First published at the Pacific Northwest Software Quality Conference 2011

1 Introduction

Kanban is a fairly new agile software development method, derived from Lean methods. Like all agile methods, Kanban insists that work must be split into chunks which have value to an end user or buyer. However, Kanban manages the chunks of work differently than many agile methods:

- Kanban doesn't insist on cross-functional teams.
- Kanban doesn't prescribe specific roles for the team members.
- Kanban uses a "pull" system which can operate with *or without* fixed iterations or sprints.

This makes Kanban work well in some situations where it's hard to see how to apply Scrum and other popular agile methods.

Our organization has had considerable success in applying Kanban. This is the story of one of those efforts.

2 What is Kanban and how does it work?

Kanban is a system for optimizing both the number of work items which can be done by a given team and the response time to new requests. The method was developed in Japan in the 1950s to maximize manufacturing system throughput. The method is called "Kanban", which means "billboard" or "card" in Japanese, because physical cards are used in many Kanban systems. Kanban and related "Lean" methods were later demonstrated to work on any system consisting of discrete items (such as a feature), activity states (such as development or testing) and wait states or queues (such as "waiting to be tested").

Kanban makes several assumptions:

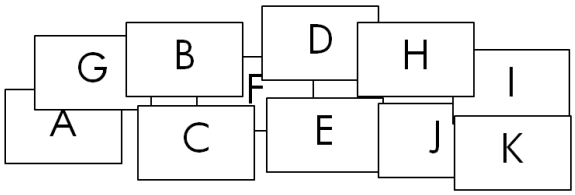
- There's a bunch of work to be done which can be split into discrete work items.
- The work items tend to arrive at different times.
- The team doing the work does not have infinite capacity.
- People get more work done when they can focus on one or a few items, rather than bouncing between many different items in the same day.

Agile software development fits into this model quite well – our work items are generally features, the customers persist in thinking up new features, we cannot do an infinite number of features, and we definitely get more done when we can focus.

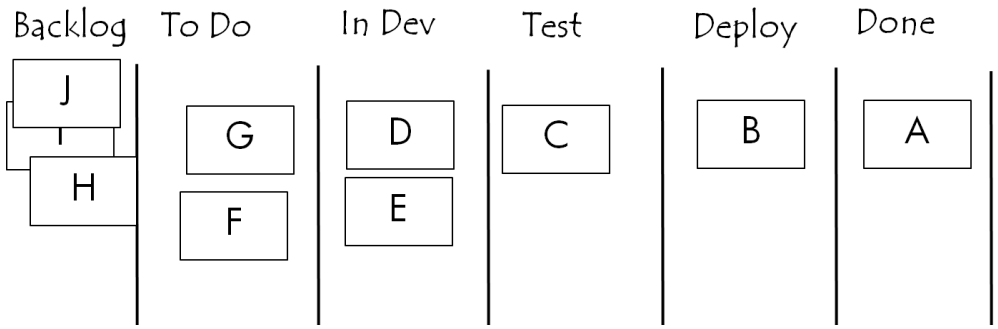
Why Kanban works can be most thoroughly understood by using queuing theory, which is the mathematical study of items moving through activity states and wait states. [REI2009]. Queuing theory itself is beyond the scope of this paper, but it is possible to understand most of the effects by envisioning your system as a set of states through which the work items are moving. In this paper, we'll use these ideas to show how Kanban focuses on maximizing the throughput of work items through a software development system, and then give concrete examples of how we used Kanban and what we discovered.

First, let's look at how Kanban allows us to focus while maintaining a predictable and speedy response time.

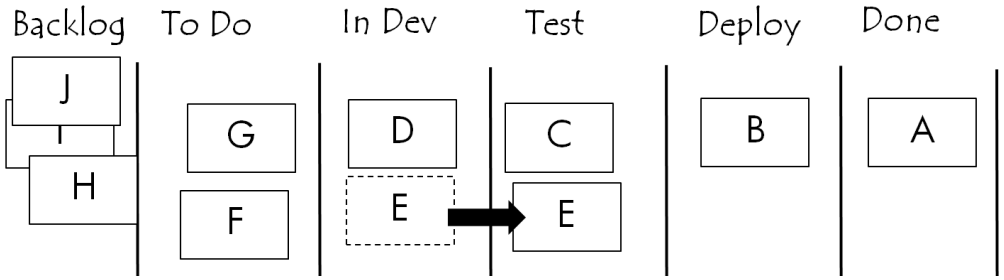
In Kanban, the work is managed this way:
1) The work is split into pieces, which are represented by cards.



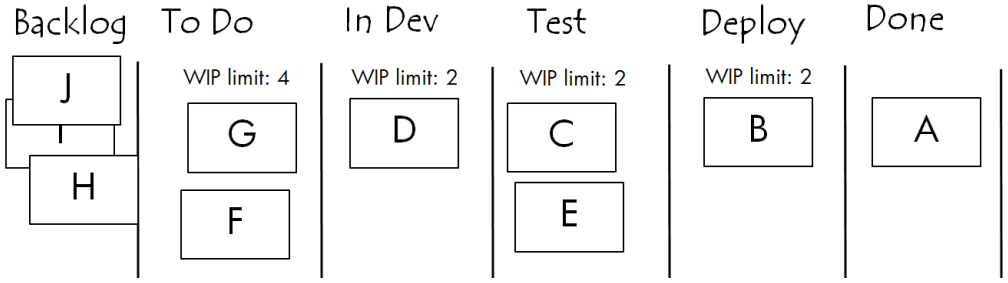
2) The progress of the work is visually tracked through the necessary steps or states, using the cards. This display is visible to all team members (and any other interested parties) at any time.



3) Work is "pulled" through the system – that is, new work items are pulled into each step by the people who are responsible for that step.



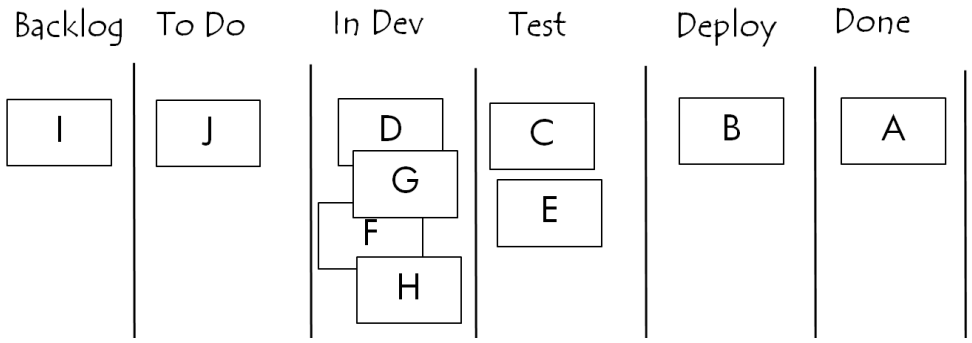
3) The flow of work through the system is controlled by strictly limiting how many work items can be in each step at the same time. This limit is known as the *Work-in-Progress* or *WIP* limit. New work items can be "pulled" only if the step is below its WIP limit. In the situation below, "D" cannot be pulled into testing because Test is already at its WIP limit.



This likely will leave one of the developers idle. Instead of pulling one of the "to-do" items into development, the developer is expected to go help the test team until the number of items in Test drops below the WIP limit.

So, why does this create a predictable and fast response time? There are several reasons:

- The strict WIP limits allow the team members to focus on just a few work items at a time. This allows the same number of people to do more work (with less stress) because they aren't wasting time on task-switching.
- The progress of work is extremely visible. If one step has gotten "stuck", it will be obvious to the team in two ways: the step has stopped pulling items from the previous step, and the next step will not be able to pull any new items because nothing is ready to be pulled. People in both the previous step and the next step will quickly run out of work.
- The WIP limit forces the team to fix the "stuck" spot, instead of continuing to pile up work in the system. Without a WIP limit, if testing was stuck, development would continue to take in new work, resulting in a situation like this:



The developers are busy, but no new features are getting to deployment because the test group is overloaded. More development will not fix the problem – more testing is needed.

In Kanban, if one step is at its WIP limit and the upstream step has finished one of its items, the item cannot be pulled into the "full" step. Instead, the upstream people are expected to go help with the step that is at its limit. In this case, the developers are expected to go help the test group until a spot opens up in the test queue.

This seems counter-intuitive to many people, because the developers will (usually) not be able to test as fast as the testers, not being as familiar with the tooling and so forth. However, the overall throughput of the *entire system* is higher – even though some individual people aren't working at their personal top speed. This is provable using queuing theory, and has been demonstrated in practice in manufacturing repeatedly for the last few decades.

There's a more detailed set of pictures of how Kanban works in [KNI2009], which is readily available on the web.

3 A more detailed look at Kanban

The first step in Kanban is to define work items or chunks of work. The same rules as used in agile development apply, but envisioning your development system as a set of states does make some things a bit easier to decide.

The first issue to consider is the type of items which can be treated as a chunk. The point of Kanban is to optimize the system to produce saleable stuff, not just do work, so the work items *must* have value to an end user. Chunks can be stories, minimum-marketable-features or anything of that sort. Architecture documents, investigations, and partially finished code are not considered legitimate chunks.

The second issue to consider is the ideal size of a chunk. The chunk is acting as a batch moving through the system. If the chunks are too small, the overhead of dealing with each chunk (for instance, entering it in the Kanban system, and prioritizing it) becomes too large a fraction of the total work being done. If the chunks are too large, the response time to customer requests becomes very long. (For more information, see the discussion of "transport cost" and "holding cost" in [REI2009]). Fortunately, it's not necessary to hit the exact ideal batch size to get fast, predictable throughput. Most organizations try out some sizes until they are getting acceptable throughput.

In Kanban, the chunks start out on a backlog. This backlog isn't prioritized in a 1 to N order. Instead, a group of stakeholders meets regularly to decide which chunks should be started and those are marked as the next ones to be pulled.

Once a chunk is "in progress", it moves through whatever states are needed to accomplish the work in this particular process. The team defines the states based on its normal workflow and handoffs between different people. For instance, if one group of people develops a change and a different group deploys the change into production, there would be two states – "Develop" and "Deploy". The team maintains a *Kanban board* showing, for each chunk, its state and who is working on it. Co-located teams usually have a physical board with Post-it® notes for each chunk, while geographically dispersed teams need an electronic board. (See Appendix A for a sample Kanban board.)

As described earlier, WIP limits are maintained on each state as well as the entire board. The developers aren't allowed to push a story into deployment if the deployment group is already at their WIP limit. Since they can't push a story into deployment, they also can't pull another story off of the backlog and start it, because that would overrun the development WIP limit. The development team's only remaining option is to go help the deployment team clear their queue, which is known as "swarming". The result is that the flow of work through the system is maximized – the largest possible number of chunks is accomplished in a given time box.

As in Scrum, there is a daily standup meeting. In Kanban standups, instead of querying each team member about what they are doing, the team "walks the Kanban board". Starting from the state closest to "done", the team looks at each chunk, asking "is this progressing as expected? Does anyone need help to get this chunk done?". There are no progress reports, because the Kanban board is visibly showing the progress.

4 Our experiences with Kanban

4.1 Who we are and what do we do

The Core Test Strategy Lab (CTSL) is a system integration and test lab for LaserJet enterprise systems. Hewlett-Packard (HP) releases a large number of LaserJet products each year. These products consist of a great deal of sophisticated software, firmware, hardware, and allied web services, which are produced by dozens of individual teams. The Core Test Strategy Lab provides system integration and testing at the system level for these solutions. The lab altogether comprises around 100 people.

We started managing all our work in Lean fashion January, 2010, and three of our teams adopted Kanban in August, 2010. The three teams are respectively

- designing and developing reusable tests
- creating and maintaining a bevy of small tools for use within our lab and maintaining our QualityCenter projects
- designing, implementing, and executing very large-scale tests in our Enterprise Test Lab.

4.2 The Tools Team Learns Kanban

In the summer of 2010, the CTSL Tools Team was searching for a means to accelerate their work. High demand from several stakeholders made prioritization difficult. In order to get work serviced, stakeholders began requesting work at the last minute and setting near impossible delivery dates. Several requests were given to the team on the same day the deliverables were needed. Resources felt overloaded and unable to get enough time to deliver key enhancements to the organization. Due to the lack of time, plenty of technical debt had been inserted into the tooling, making small updates and maintenance continually take more effort than expected.

Kanban was presented as a possible solution to the tools team by Kathy Iberle. She learned of its uses at the 2010 Lean Software and Systems Conference[1] , and afterwards introduced it to the lab. We ordered a copy of David Anderson's book, *Kanban: Successful Evolutionary Change for Your Technology Business* [AND2010], for each member of the tools team and required them to read the first several chapters. Then we decided to try Kanban, but we needed to decide how to implement it. The tools team is remote, so each member was flown in to Boise, Idaho to discuss the process. An entire day of the offsite was dedicated to Kanban training, led by the manager of that team.

After the day of theoretical training and answering concerns, the tools team discussed how they would implement the process to begin a pilot. The team started creating its own Kanban board by deciding on WIP limits for each column. This decision is an important one for Kanban. Setting the number of slots is Kanban's method for reducing WIP and task switching. Any columns with too many slots create opportunities for work to pile up and not get completed. It also eliminates the ability for teams to collaborate and work together on projects. The tools team purposefully dropped our limit such that every person could not have more than two independent chunks of work in the system at the same time. We decided to have a WIP limit of ten in the under development state and committed to a WIP limit of seven in the committed/input queue. This queue would replenish weekly in a joint meeting with stakeholders, where the stakeholders advocate amongst themselves on which requests should be committed to each week.

[1] The LSSE conference is produced by the Lean Software and Systems Consortium http://www.leanssc.org/

4.3 Day to Day Kanban

In order to understand the benefits of using Kanban in the tools team, it is important to understand how the daily activities of a developer on the team are driven by the process. At least three times a week the day begins with a standup meeting where the team reviews the Kanban board. The board is covered from right to left, first discussing Kanban cards in the done state, then under development, and then finally what is in the selected queue. Work is discussed that has been stuck in a state for long periods of time, looking for opportunities to swarm and briefly brainstorming on possible solutions. Unless work is stuck in a state, current status is not shared.

There is no time spent on Kanban cards in backlog. Only work that has been committed to gains dedicated resources. If a developer completes one requirement, the developer checks first with any blocked team members if they need assistance before pulling from the committed queue. This drives focus and attention on finishing work before starting new work.

4.4 How work gets prioritized

At any given time, we're serving half a dozen different programs, each with its own stakeholders. We use Kanban's method of prioritizing the work at an Input Queue Replenishment meeting. This gives all the stakeholders a voice in what to do next, in a quick and organized fashion. As the managers of the teams use Kanban, they learn the capacity of their team (similar to agile "velocity") and can avoid overloading the team, which would cause slow response time.

The Input Queue Replenishment meeting ensures that each requirement coming into the tools team is agreed upon as a top priority by all necessary stakeholders. This required the stakeholders to decide which problems to focus on, which was an excellent means to limit work in process for the tools team. Each item placed in the queue has an agreed-upon value, timeliness in its need for delivery, and an actual or surrogate end user ready to verify and provide more specifications to the tool developer. If there wasn't a consensus at the Input Queue Replenishment meeting or there wasn't a solid agreement, the work item doesn't even make it into the "committed" queue.

4.5 The Kanban Board

Because all our teams are geographically dispersed, we're using electronic Kanban boards. We are tracking the work chunks as QualityCenter "requirements" and using a custom-written Excel add-in to display the Kanban board (See Appendix A). There are numerous Kanban-board tools available commercially today, but our IT department is not supplying any of them as of this writing. (Many of them store the data outside HP firewall, which disqualifies them for obvious reasons). It is possible to manage a Kanban project directly from QualityCenter, by representing all the chunks as requirements, but displaying the information in the classic Kanban board format certainly makes it easier to see what is going on.

Most authors recommend that the Kanban board be publicly visible on a wall, in the style of agile "information radiators". We can see the value of having our progress visible to our stakeholders all the time, but we haven't yet developed the technology to make this possible.

4.6 Benefits of Kanban

The tools team saw immediate benefit from the shift in day to day activities during the Kanban pilot. The WIP limits per queue allowed for strict capacity control without discouraging collaboration. The Kanban board itself brought to light the length of time relatively small changes were taking, driving buy-in from stakeholders to allow the tools team to pay some of its technical debt. The weekly meeting to insert work into the team's "committed" queue eliminated the last-minute requests. Once the maintenance and minor changes were in control, the team was able to begin addressing new feature requests. The developers did not feel overwhelmed or overworked, since regardless of the backlog size only a certain amount of work was going to be committed to each week. The turnaround was noticeable from stakeholders as well as developers in just one quarter, and afterwards other teams in the lab were looking to implement Kanban to reap the benefits.

We found that just having a Kanban board has returned time to the developers. It has removed the constant "status checking" from managers for the developer, as well as the lag time in getting feedback for the manager. Any stakeholder, partner, manager or interested individual can instantly get an understanding of the team's scope of work and what it currently has in its system by checking the Kanban board. There is a reduced need to request status e-mails, status meetings, and even how the daily standup operates. The tools team only discusses work items that have remained in the queue for an abnormal amount of time, or where the tool developer needs assistance from a peer in its status meetings. This part of the process drove the average length of the team's standup meeting down significantly, from 1.5-3 hours per week on status to about 45 minutes to an hour per week. Expanding this to a team of six developers, the reduced standup time provides 4 to 12 hours more per week to focus on completing the work items in the queue rather than reporting on their status.

Having each team member's work in one easily read location also provided some less tangible benefits. Collaboration has greatly improved among the developers supporting different tooling applications. This drove to more consistency in tooling, work styles, and a better understanding of the organization's strategy and the tools team's place in it.

One example of this is in the performance of our tools supported by different developers. Before Kanban, our test setup application was notoriously slow, taking on the order of 30 minutes to pull all the necessary information down from our central test library. Our metrics reporting tool could pull the same data and even more in approximately one minute. One day during a standup meeting, this discrepancy arose as the tools team was asked to address the test setup application performance. Finally made aware of the size of the discrepancy, the metrics developer assisted in porting his code for tool setup application usage. By the time this performance work was completed, the test setup tooling could pull the required data at the same speed as the metrics application. Collaboration like the example above can certainly occur without Kanban, but the forums and inclusiveness of Kanban created more space for such cross-functional discussions.

Kanban has driven the length of our work tickets to a somewhat consistent size. Larger projects are decomposed into minimum marketable features, and are developed iteratively with rapid prototyping of the defined solution so feedback can be gathered from the involved stakeholders. These techniques enable shorter engagement periods for stakeholders, allowing them to focus on more than tooling design while still providing enough direction for the tool to make a maximum impact.

5 Situations in which to consider using Kanban or not:

5.1 Places where Kanban works well:

- Relative priorities of chunks change often. When priorities are changing faster than items can be pulled off the queue, updating a sorted 1 to N list will be wasted effort.
- Desires of multiple product owners have to be balanced off against each other. In Kanban, stakeholders are told they will meet every week to collectively choose however many items can be started that week (typically a small number such as 3). This reduces the stakeholders' job in any given week to picking the 3 most important items to start that week, rather than engaging in a protracted battle over many weeks to prioritize 100 items according to the stakeholders' varied interests.
- There's no obvious cadence for "sprints" or it's not obvious what a sprint would consist of. Sprints in Scrum perform several purposes – control WIP, provide a cadence for integration and deployment, and provide a cadence for stakeholders so they can change their minds as often as needed. Kanban controls WIP with its explicit WIP limits instead of with sprints. There is a cadence for stakeholders, provided by the weekly stakeholder meetings.
- You are tired of tracking tasks and effort hours. Kanban doesn't require tracking either one. Team velocity is measured in chunks. When initially created, chunks are usually sized into Small, Medium, and Large, and some care is taken that extra-large chunks are broken into smaller chunks, just as epics are broken into stories. Since the team takes on work according to its WIP limit, its average time to process a chunk becomes fairly predictable. Once the average is well known, most stakeholders will use that average in their planning rather than demand an estimate for every chunk.

5.2 Places where Kanban will not work well:

- The work items need to be integrated together *with each other* before being deployed. Kanban doesn't have a good way to handle the grouping. Fixed time boxes or iterations with an integration at the end of each iteration are a better approach.
- The organizational structure is highly cross functional, such that the steps to complete a chunk alternate between teams using Kanban and teams not using Kanban. Two different prioritization methods in use on the same items will cause a lot of delays and frustration.
- Firm commitments must be made well in advance of the delivery dates and last-minute changes in priority are not allowed. The frequent re-prioritization in Kanban would not be necessary, although the rest of the method will probably work.

6 Conclusion

We've found Kanban to be an admirably lightweight method of tightly managing both capacity and throughput of work in an environment of rapid change. It's easier to manage our teams' workloads, see when collaboration would be useful, and avoid working on low-priority items. When we switched from Scrum-style standups to the Kanban-style standups, we found that our standups got shorter, the team members were better able to help each other, and the meeting was more energetic. The work tickets are

converging to a consistent size and rapid development of prototypes enable stakeholders to remain engaged in more than just the input queue replenishment.

References

[AND2010]: Anderson, David J. *Kanban: Successful Evolutionary Change for Your Technology Business*. Sequim: Blue Hole Press, 2010. Print.

[KNI2009]: Kniberg, Henrik and Skarin, Mattias. *Kanban and Scrum – making the most of both*. InfoQ, 2009. Web. 10 July 2011. <http://www.infoq.com/minibooks/kanban-scrum-minibook>.

[REI2009]: Reinertsen, Donald G. *The Principles of Product Development Flow*: *Second Generation Lean Product Development*, ch. 3. Redondo Beach: Celeritas Publishing, 2009. Print.

7 Appendix A