State of the art of Agile methodologiessss

Michael Shell
School of Electrical and
Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Email: http://www.michaelshell.org/contact.html

Homer Simpson
Twentieth Century Fox
Springfield, USA
Email: homer@thesimpsons.com

James Kirk
and Montgomery Scott
Starfleet Academy
San Francisco, California 96678–2391
Telephone: (800) 555–1212
Fax: (888) 555–1212

***Abstract*—The abstract goes here.**

I. Introduction

This demo file is intended to serve as a "starter file" for IEEE conference papers produced under LaTeX using IEEEtran.cls version 1.8b and later. I wish you the best of success.

mds
August 26, 2015

A. Subsection Heading Here

Subsection text here.

1) Subsubsection Heading Here: Subsubsection text here.

II. Kanban

A. Origins

Kanban, as a method for managing and transmiting knowledge, was invented in 1953 by Taiichi Ohno as part of the Toyota Production System (TPS), which later was adopted by other companies and is regarded as one of the most influential systems, if not the most, behind the Lean Manufacturing philosofy. As a methodology developed for a company - Toyota Motors - operating in the war-devastated Japan and competing against well estabilished US companies, TPS focused on reducing overburden, inconsistencies and waste. In order to achieve those goals, a set of guiding principles, which became known as *The Toyota Way*. Some of those principles are:

- Create a continuous process flow to bring problems to the surface
- Standardized tasks and processes are the foundation for continuous improvement and employee empowerment.
- Use visual control so no problems are hidden.
- Use pull systems to avoid overproduction.
- Level out the workload (work like the tortoise, not the hare).

While the first two are clearly lean principles, the other three will be revisited later in this section, as they are essential for understanding the current Kanban. Central to this notion of eliminating waste is Toyota's Just in Time (JIT), where a component is only produced when it is needed by a station further beyond in the production pipeline. So Kanban was invented so that there was an easy way to spot innefiencies in the material flow of an assembly line, such as bottlenecks and waste (in this case, that would be having unused materials, or inventory).

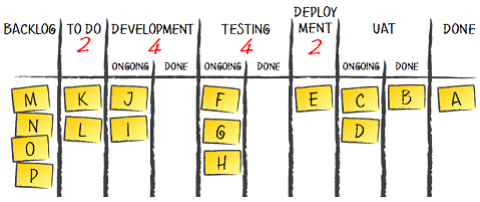

Fig. 1. Example Kanban for software development.

B. What is it

Kanban is japanese for signboard (*kan* means card and *ban* means signal), and in the TPS it's just that: a physical board that workers write on and move around which they use to communicate material requirements. When a station needs a certain material, they fill a Kanban and deliver it in the station that produces it.

However, Kanban has evolved to be more than that, and is currently not restricted to visualization, but is regarded as a lightweight framework for project management. The way it works is the following: you have a board with several columns, each representing a stage of the process. Then, you add to the first column the tasks you are starting, and move them to the next column as soon as they enter that stage of the process. In figure 1, one of example of this mechanic would be: task J is under development, and as soon as it is considered *Done* it can be moved to testing. It may seem too simplistic, but we have to keep in mind Toyota's Way *Use visual control so no problems are hidden*. There is more than meets the eye though, because as we already stated before, Kanban is more than visualization. Below we outline Kanban's core principles. We will elaborate on how to use a Kanban effectively throughout this section.

1) Flow instead of batch: Unlike time-boxed methods like Scrum, where the whole planning orbitates around a sprint, in Kanban there is the notion of continuous flow: tasks keep comming in and being delivered, teams don't have to wait until the end of the sprint to deploy the changes. This idea is highly compatible with lean principles and continuous integration practices.

2) Definition of done: As in other agile methodologies, such as Scrum, it is pivotal to have a clear and strict definition

of done, since tasks can only move from one column to the next when those conditions are met. This definition is stage-specific, meaning that in our example there must be a definition for Development, Testing, Deployment, etc. Example definitions for our Kanban in figure 1 could be, respectively: *"Satisfies user story or fixes bug"*, *"Passes all tests, including regression ones, and has at least 60% converage"* and *"Is deployed in our production servers and users have access to it"*. The absence of such definitions decrease a team's velocity, since its members would have to stop being productive to wonder, or discuss, if a certain task should be moved to the next stage. When a task is considered Done in a certain stage, it is passed into that stage's sub-column, ready to be pulled (we will explain what this means in the following paragraph).

3) Pull instead of push: Tasks are not assigned to a person, but instead each person will grab a task for a number of different ones available, as long as they're in the first column, or in a done state. It's like going to the supermarket: the products are there for you to chose from. Even if Toyota's *Use pull systems to avoid overproduction.* may not seem to apply to software development, it is still used in Kanban. The rationale behind it is that it gives people autonomy, thus increases velocity as well as satisfaction. It's a reflex of Toyota's *Respect for people*. In software development, this practice can manifest itself in several ways. For instances, developments teams do not hand out their code for QA, instead QA teams will pick *Done* features and start working on them. It can also impact DevOps/Release teams, where the teams which handle deployment aren't waiting for development to hand them the production-ready code, what they do is create the necessary infrastructure for any developer to be able to put their code into production without much trouble. This is how engineering teams work at Spotify. Again, notice how lean and close to continuous integrations this practice is.

4) Work in progress: If tasks are picked and not assigned, there is the risk of stalling the system, having many tasks in the pipeline, but few in the final stage. It can also happen that team members decide to pull too many tasks for themselves, losing productivity to multi-tasking and context-switching. To avoid this kind of problem, as well as maintain an healthy flow in the system, Kanban uses limit the Work in progress (WIP). WIP means that there is a certain capacity associated to each column (besides the first and the last ones) and that you can't pull more tasks in a column that is already full. Both ongoing and done tasks contribute to reaching the WIP's capacity. In figure 1 you can see WIP red numbers. Each team must find the balance between having a low WIP, so they avoid the problems outlined ealier, but not so low that it starts getting in the way of productivity. Finding the right WIP is highly specific to each team and stage. For instances, it may make sense for a team of 5 developers to have a WIP of 10 in the development stage, but it would be unproductive for the same team, if they have a single tester, to have the same WIP in the test phase.

5) Estimation, prioritization and delivery times: A criticism made to Kanban, mainly for people with Waterfall or Scrum backgrounds, is that predictability is low. Since the methodology does not enforce estimation, there is no way to track velocity or to predict delivery timings. There are two answers for this: the first is that you can actually have a reasonable way to track velocity, if that's what you really want, and the second is that you don't need to track it. For the first case, if you measure how many tasks you complete for a given time window, you can have an estimate of your team's velocity. Even if each task may have different weights, in the long term you can get a rough estimate. The reason why it is not important for Kanban to measure velocity, is that it focus on the long term: remember Toyota's *work like the tortoise, not the hare*. By having a continuous flow of work while giving ownership and autonomy to your team members, you are assuring maximum productivity for each task. So the estimated completion time for a task would be as soon as possible. Have in mind that, by being a lightweight framework rather than enforcing a set of strict rules, like for instances Extreme Programming does, there is space to modify Kanban for your own context. You can for instances use Kanban in conjunction with Scrum, and model a sprint in terms of a Kanban board. More on this in the next section.

C. Relating to other methodologies

1) Open Allocation:

2) Kanban with Scrum:

3) Kanban instead of Scrum:

D. Testing

III. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.