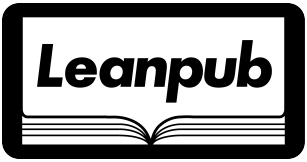

Holy Land Kanban
Best of the Agile/Kanban
Blog from Israel
By Yuval Yeret

Holy Land Kanban

Best of the Agile/Kanban Blog from Israel

Last published on 2012-02-27



This is a Leanpub book, for sale at:

http://leanpub.com/holylandkanbanbestof

Leanpub helps authors to self-publish in-progress ebooks. We call this idea Lean Publishing. To learn more about Lean Publishing, go to: http://leanpub.com/manifesto

To learn more about Leanpub, go to: http://leanpub.com

Contents

0.1 Acknowledgments

I'd like to thank all my blog readers and commenters who provided me with feedback and kept me going, my colleagues in AgileSparks who provide me with the sandbox to try my crazy ideas and my family - especially my wife and kids who bear up with the time spent on writing and thinking.

1

Kanban - Beyond the Basics

1.1 Collaborating with specialized roles using kanban classes of service

I want to share a solution I came up with together with a team of performance / non-functional testing, working in a product group in a large enterprise. This solution deals with the challenge of bridging the principles of "Those who build the system test it", "Non functional testing is a collaboration role", and the fact that specialized roles such as performance testers are usually stretched to cover the demand for their services.

This group wanted Performance testing for the product of course. What usually happened though is that the performance team only got usable features towards the end of the release (think waterfall like behaviour). This usually meant serious waivers and compromises around performance testing.

The first step taken by the product group was to work more on effectively breaking the features into smaller viable features and stories. Once this was done, it was easier for the performance testing team to get involved throughout the version, and achieve a more reasonable flow.

Things should have been great by now.

BUT then another problem surfaced, even while we were discussing how this would work.

It was clear that the capacity of the performance testing team wasn't sufficient to really address everything.

The naive policy meant that when a feature arrived at performance testing, they would decide whether they have time to

cover it, do risk management and either test it or skip it.

The downside for this was that its a black/white decision. This misses the opportunity for the delivery agile teams to do at least SOME performance testing, even if they don't have the capabilities, tools, expertise of the dedicated performance testing team.

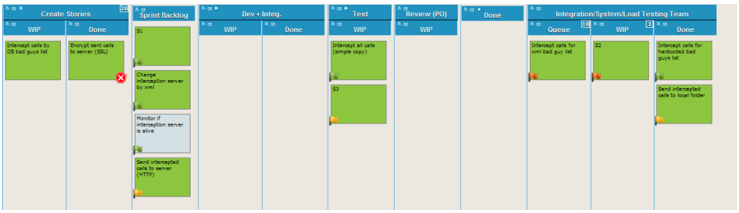
Our suggested solution was to use the concept of kanban Classes of Service to improve on this naive policy.

Since we already know not every feature requires the same performance test attention, lets not wait until it arrives at the performance team to make this classification. Lets do it earlier, before we go into development/testing.

With this classification, policies can be setup that can involve both the performance testing team as well as the delivery agile teams in the work of performance / non-functional testing.

We came up with a flag system:

- Red – performance team Must be involved hands on – probably by joining the Feature team working on this feature for the duration of the effort
- Yellow – performance team Advise/Consult, but most work is in Teams. Representative of the performance team will be visiting the Feature team quite often while the Feature is being worked on.
- Green – don't need any involvement from performance team



Classes of Treatment in Kanban

This system helps drive collective ownership of non-functional testing. One of the first things that happened is that the Feature teams told the performance testers that there are some kinds of tests they can run on their own, although they don't have highly specialized performance tools.

We are also experimenting with systems like this for involving architecture teams, and it applies for most kinds of super-specializations that are not easily integrated into Feature teams.

Managing the overall flow using kanban will also help see whether a bottleneck develops, and what can be done further to overcome it.

1.2 Encouraging Feature-level progress tracking in Kanban

One of the key questions project managers and senior management in general ask themselves and their teams on an ongoing basis is - "Are we on track to deliver the scope we committed to, on time". In some environments "on budget" is added to the question.

If you are talking about a Release Scope, the answers are quite similar whether you're doing Scrum or Kanban. If you don't care too much about the budget aspects, a Release Burnup can show you the commited scope, the committed date, and the actual progress in working software towards that goal – Plan versus Actual. If you ARE interested in the budget picture – committed budget versus actual, and are we on track to finishing the release with the budget we committed to – use AgileEVM[1] on top of that.

Basically for all of this – you are measuring the amount of done features work compared to the amount of features work originally planned for. Whether sized using effort days, story points, function points, the idea is the same.

In a conference a couple of months ago I talked about Agile Release Management[2] and covered this subject somewhat. I would add that this expectation of management is what we call Predictability in the Kanban world, and based on some

[1] http://www.infoq.com/articles/agile-evm

[2] http://www.slideshare.net/yyeret/managing-projectsreleases-using-leanagile-techniques

encounters I've had with senior management, we as the Agile community have not been doing a great job at connecting to the expectation of Predictability. In many cases its the opposite – we create the impression that Predictability is a lost cause because everything is Agile.

In Kanban we try to better connect to this expectation of Predictability/Commitment to the important things. Senior management doesn't care about committing to a sprint goal and meeting it. They care about meeting commitments to deliver a release on time and with feature highlights communicated to the stakeholder community. They care about meeting commitments to deliver certain features on time to internal and external parties that count on this feature to continue and do something else.

Predictability will continue to be important. The way its measured might change. For now, most teams/projects are indeed evaluated based on the answer to "Are we on track to meeting the release goal on time". We should support those teams with an approach that complements their kanban flow-based workflow. The methodology is all there if you connect the dots.
The room for improvement is mainly in connecting the dots and providing a structured methodology that can be applied as a framework, as well as better tool support.

What are the gaps?

First, The thinking around CFD needs to switch from history to also a forward-looking predictive chart. What do I mean?

Most CFDs you see today focus just on an operational view CFD – what is the current state, as well as history, which can help you improve your process, operation.

I'm Missing a view of the work needed by a certain date, and

whether we are on track to achieve our commitments/goals. Tools that extend the CFD to a view that includes current trend, required trend to meet the goal, and trend of requirements churn can answer this question – you see whether the DONE trending towards the overall committed scope is on time or not.

One more complication is that of course you sometimes want your board to reflect many releases, not just one. You're working to finish one release, and then you move to another.
In this case, You probably want this view per-Release on the board.

So we need visibility charts that can aggregate the status of several cards e.g. Feature, Release, MMR, MMF whatever you want to call it. In Feature Driven Development[3] Parking Lot diagrams[4] are a popular way to convey the status of various features/aspects in a Project/Release. An extension of a Parking lot diagram can be to have a mini-burnup of that entity. So beyond just the status (which is basically the current point of a burnup), you can have a mini-graph showing the status of entities comprising this feature. See below for a sketch of how this can look. (Note that the Warning Indicators box are taken straight from the organizational dashboard page of LeankitKanban. I recently started to explore the capabilities in this dashboard and find them quite useful to help bring a process under control, and the sort of stuff you might want to look at in an operational review).

[3] http://www.featuredrivendevelopment.com/

[4] http://www.nebulon.com/articles/fdd/ptsparkinglot.html

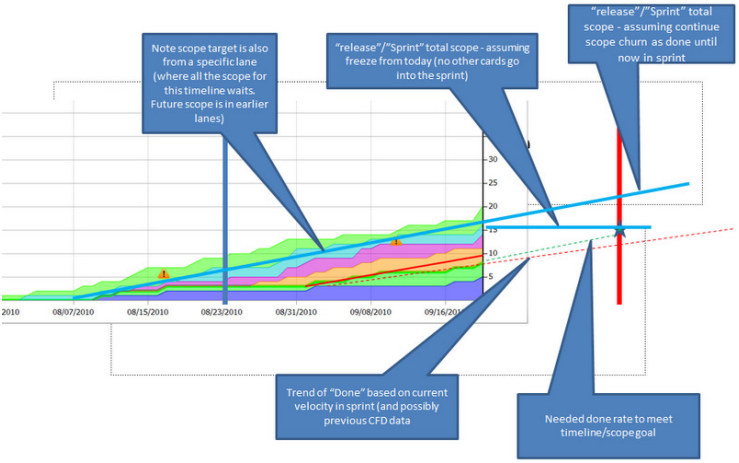
Parking Lot Charts

The color of each parking lot / feature can easily be derived from where the actual progress is compared to the expected progress curve. The expected curve can be defined to be Linear (yeah right), S-curve based as David Anderson is fond of[5], or whatever you think the profile should look like. Once you are below the curve, you start to gain reddish colors. Above it – you are green. With Agile approaches relying on Working Software as a measure of progress, you can really trust those colors… Its no more a watermelon (green outside, red inside – courtesy Kent Beck[6])

For those interested in the details, here is one way a CFD can be extended to provide burnup capabilities.

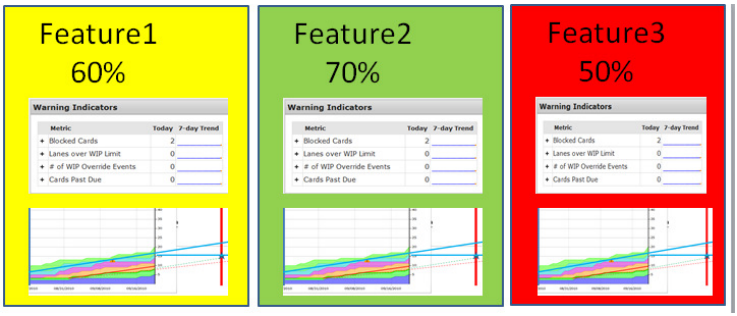
[5]http://edn.embarcadero.com/article/32411

[6]http://twitoaster.com/kentbeck/rt-pmagile-a-little-joke-i-told-at-the-agile-conf-execs-see-watermelon-projects-green-on-outside-red-on-inside/



CFD with Burnup Trending/Forecasting

With this in mind, the mini-burnup in the parking lot can be upgraded to a mini-CFD



Parking Lot Charts with a Mini-CFD

Now, with a CFD, some more intelligence can be applied to help

determine the color/state of the Feature. High level of WIP can be a leading indicator of problem (but knowing about Little's law and how a CFD looks like you probably know that it will be apparent in the burnup being quite flat as well…). I'm guessing that with time, we will learn to study and identify progress risks using a CFD, beyond the basics we currently use.

Bottom line – my feeling is that in order for Kanban to cross the chasm into the majority of projects/development groups, who are quite concerned with delivering Releases and Features on schedule, not just with trusting the Flow, we will need to provide more and more tools and focus to support this use case. The core thinking is there, the hunger on the part of the IT world is there as well it seems, so lets go out there and make it happen. my 2c…

1.3 How I would simulate time-boxed agile in the @getkanban kanban board game

At Agilesparks we've recently been using the Kanban Board Game [7] developed by Russell Healy quite extensively. We use it as part of Kanban courses, sessions for Scrum teams that want to learn about Kanban, Kanban teams that are already working and want to raise their game as well as in sessions for project management teams.

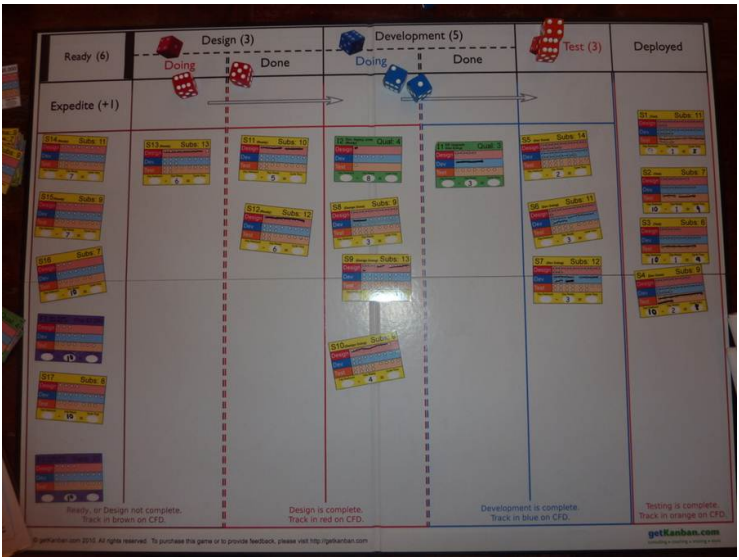
We love what the game does, and it has actually become a challenge to coordinate which of us has the game kits at any point in time, especially because in large events we need 4 kits so we need to borrow from one of our nice clients that has 2 kits as well…

One idea we've been toying with for some time, is that it would be great to use something like this game to accelerate learning of Scrum as well. At a minimum, one of the questions I'm adding to the debrief for Scrum teams, is how would Scrum look like in this game, and what would be the effects. It seems clear to teams that Scrum would reduce throughput due to the need to clean the board every sprint, which means less ability to take advantage of the speciality of the Analysts/Developers/Testers. It would also make the game run slower due to the overhead of planning…

I've been reading some ideas from David Anderson and Russell about this, which helped me along. I'll provide my take on this,

[7] http://getkanban.com/

but first a brief overview of the game to those not familiar with it.



The GetKanban Game

The game simulates a process workflow where there is a stack (backlog) of cards which represent stories for development. The cards are pulled into a READY area, then pulled into Analysis, Development, Testing and finally deployed. There are different types of cards representing different work types/classes of service – a normal business value card, a fixed date card, a quality card, and an expedite card. Each card specifies the amount of work required to process it in each station, which provides the variability between work items so typical of product development and other knowledge creation areas. Another form

of variability is the fact that how much work you are able to do each day is decided by throwing dice. You have dice in different colors that represent specialists in each area (Analysts, Developers, Testers), and you as the team playing the board can decide where to use each dice each day. Either use it in the specialty area of that person and gain a multiplier, or use it elsewhere without a multiplier. The game is played in cycles of days – each day you decide your strategy, throw the dice, play the game, update visibility charts, and also handle an event that might throw you some additional curve balls… That should suffice as a quick intro to understand the rest of the post, or at least as a teaser to find the opportunity to play the game. It really is highly recommended to anyone interested in Agile / Kanban / Scrum / Games. Exceptional work by Russell.

If you noticed so far, the game indeed simulates kanban. No time-boxes or batches, just flow and flow and flow. The only area where you get some sort of sprint behaviour is towards the end of the game. If you are approaching the end of the game and the teams know its coming, they will start to prefer cleaning the board and deploying items, rather than maintaining flow. That makes sense. If you announce end of the game on day 24 for example, but finish it a few days early without telling the teams, you will get flow behavior all the way to the end. You can play with that, depending what you want the teams to experience, and whether you want to have the timebox behavior as a pattern to discuss in the debriefing.

In general, it shouldn't be too hard to make the adaptation of the whole game to timeboxes/sprints. maybe the biggest challenge is to have teams see the name Kanban on their new shiny Scrum process training…

But seriously, the main thing you need to take care of is the sprints. The concept of 3 day billing cycles in the game can be extended to portray a sprint. You would start the cycle with planning what cards you will work on in the cycle. This requires looking at your velocity in the last cycle, your capacity (cubes and statistics…) thinking about changes that were made in the environment (Carlos etc.), and the cards that are currently in the backlog. You would commit to what cards you plan to deliver.

Which gets us to the backlog… The trivial option is to make the READY area the backlog, and only pull into the READY area (into the board actually) cards that fit into the planning. This is not perfect though. For starters, how do we model the work that is done by the Product Owner? In addition, the whole flow from READY through analysis, development, testing might be a bit too long for a 3 day cycle.

That brings me to an alternative I've been thinking about – make the Analysis phase the work of a "Product Owner". Use the analyst dice to represent the Product Owner, and the "Analysis Done" area the "Sprint Ready" backlog the teams can use for their planning.
This has the benefit of portraying the Product Owner work, it reduces the cycle time within a sprint which I think will provide a more reasonable simulation within a 3 days cycle, and can also bring about a discussion of the wastes of too much sprint ready backlog, versus a team that runs dry during a sprint. (Which is one of the common problems with Scrum teams, and a big benefit of doing this game with them I think)

So assuming the Analysis Done was the sprint candidate/ready backlog, the team does planning, pulls the planned stories into another queue between Analysis Done and Development in

progress, which we can call – the Sprint Backlog

A sprint will be 3 days where the team works to finish and deploy all the cards in the Sprint Backlog – processing them through Development and Testing. Within this sprint the team also needs to take care to replenish the "Sprint Candidate/Ready Backlog", because remember at the end of the cycle, there is another Sprint Planning, where they will need to have enough candidate cards that they can pull into the Sprint Backlog, to avoid starvation in the sprint. To properly simulate a real Scrum environment, it shouldn't be allowed to pull cards within a sprint, because it is harder to plan within a sprint.

For advanced uses and for those Scrum people which are crying this is not fair, we can discuss a couple of options. One option is to allow pulling stories, while paying a fine for it to represent the cost of replanning. For example, you can put one of the dice on planning a story instead of executing it. Another option is to allow pulling quality cards during a sprint.

In general btw, we need to think how to portray the cost of planning. In a typical stable Scrum team, there is a cost of about 2-3 hours of planning per a 10 day sprint. This is about 4% which should be negligible in the game. Another option is to just compare the time it takes to do rounds of flow versus timeboxed rounds in the game and see what is the actual time it takes to run the game… I would expect flow to run much faster than the time-boxed version… but need to try it!

At the end of a sprint, what would happen is that you expect to have clean Sprint Backlog, Development and Testing stations, with all the cards deployed already. Any work that is not completed will probably be continued into the next timebox and should be included in the plans. I'm thinking a fine should

be included here – every card that spills to another timebox should have a few points of work added to it (e.g. if there are 12 development points, the team finishes with 6 points of work left, in the new timebox there will be 6+3 points of work)

What would happen to Cycle Times? I'm quite sure cycle times will be much higher for a team doing timeboxes. The need to keep a queue of work ready for sprints will probably add about 2-4 days of cycle time, depending on how the team balances safety of having spare stories, versus the waste of having a story decay in the queue. Since the game doesn't model a price for decaying stories, I assume teams will prefer more in the queue. The only thing that will stop them from doing that is limited amount of analysts… need to try it but I'm guessing there is a chance that the analyst will not be able to keep up, or at least we need to create scenarios that impede him from keeping up (e.g. blockers in analysis, reduced capacity of analysts, or taking stories that were analysed and cancelling them so now analysts have to rush to replenish the ready stories queue)

The event cards are another area which might need tweaking. For one, some events are unfair to introduce in the middle of a timebox because the team should know about them when planning. Need to look at them and align them with the timeboxes as necessary. Beyond that, some new kinds of events can be leveraged to spice up the timeboxes game (see above for example)

Will the effect of time-boxes be realistic enough? The concern is that if the time-box is too short, it will be hard to deliver stories and run an effective time-box. Similar to what teams experience when the story size is too large for the sprint. Basically, need to play around with it and see… I think that by removing

the Analysis from the sprint itself I reduce the chance of this happening, but maybe its not enough. Worst case can play with multipliers on the cubes some to make the team capacity higher compared to the work, or strengthen the effect of the quality cards…

The WIP limits go away in this variant I think. The WIP limit is the Sprint Backlog which is time-based, not station based. An advanced team can decide to use WIP limits on the stations to work more effectively, or we can re-introduce the WIP limits as an event card during the game. While on this matter, a question I usually ask teams in debriefing is what would happen if there wasn't a WIP limit. The answer I hear is that they would probably start more work than they can finish, and throughput and earnings will go down. QED…

This should give you some ideas how to play with the Kanban game in a time-boxed manner. There are probably other aspects that need to be taken care of, but lets be agile about it folks…

And again, thanks Russell for the great game, which probably deserves to be a platform, with endless options of what to do with it. If we could have this as a Lego kit that we can play around with, it would be amazing.

1.4 Kanban early warning using a predictive variant of SPC

A Confession. While I'm a great fan of using SPC charts to explore specific cycle times and reduce variation / continuously improve a Kanban System[8] (a great blog by benjamin mitchell), I'm only seeing preliminary results in the field with teams I'm coaching. The main reasons are lack of tooling, lack of incentive to manually manage this, and the fact that teams are not yet mature enough. I'm hoping this will change in the near future.

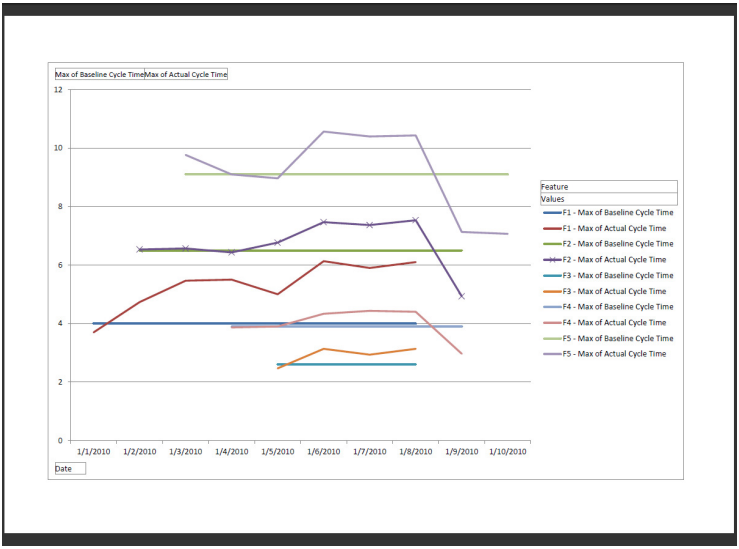
With that in mind, a constant concern I'm hearing is that finding out that there was an exception based on SPC is too late. Why is that? Because a **classic SPC looks at final cycle time**. And Final by definition means the action is over…

A couple of months ago, I read about "**average cycle time in column**" in the GSK R&D Case Study by Greg Frazer[9].

A couple of days ago, I had the idea that perhaps using the collected history of cycle times in columns/lanes, a current prediction of final cycle time can be calculated for each card in the system. This prediction can be then traced on an SPC-like chart, and exceptions can be identified more clearly (see illustration below for an example)

[8] http://benjaminmitchell.blogspot.com/2009/05/control-capability-charts-on-kanban.html

[9] http://influencecorp.com/2010/06/lean-software-experience-report-%25E2%2580%2593-rd-it-at-gsk-%25E2%2580%2593-2008-9/



Predictive SPC

This reminds me of charts used to track "Due Date Performance" on releases/milestones, which I saw at one client of ours. I later learned they are called Slip Charts[10]

The main difference is that the SPC Y-axis is by cycle time length. In the Due Date tracking chart, its by actual date. I think its probably sufficient to just rely on the SPC charts. I would urge organizations tracking due date performance on releases and milestones to start doing an SPC at that level, even if they don't use Kanban/Agile at the Feature/Story level. They might learn a few things that can help them bring their process under control, and improve their predictability.

[10]http://www.c2.com/cgi/wiki?SlipCharts

Back to the predictive SPC – no tool that I'm aware of currently offers this capability, but one can always hope.
I see capabilities such as identifying struggling work items based on exceptions from "average time in lane" as well as overall exception in predicted cycle time key to taking the "early feedback and action" one step forward, and scaling Kanban to be something project managers swear by.

If you are aware of any of the Kanban tools that provide this – I'll be happy to hear about it.