Converting a Scrum team to Kanban

Mattias Skarin

Crisp

1. Abstract

In 2009 I met a team in trouble. They were working big amounts of overtime and caught in an evil code-code-don't ask loop. Their mission was to replace a core business system for a key client. With the deadline two and a half months away (the client putting funding and future engagements on ice) the challenge was to surface the right problems and apply countermeasures that would have rapid effect.

This paper tells how we used Kanban as a methogical approach to surface and solve problems. Kanban helped us:

- Surface the impediments preventing flow of value
- Bring together line managers, project managers and developers to overcome them
- Step by step shift the focus from coding to high quality deliveries
- Maintaining a dicipline of quality, even in times of high pressure
- Bring surrounding parties (such as the client developers) onboard with the changes we made

The delivery was made in time and the client came back to the company with their next project. At the time of go live, the team was no longer working overtime. During the period, the velocity if the team increased during with a factor of 1.9.

Converting a Scrum team to Kanban

Using kanban as a base for continuous improvement

Intro

The team was I trouble. They were six month into their 8 month project when the client threatened to pull the plug. This meant saying good bye to a key client. What had initially run well - a pilot delivered on time using, a happy client, had turned into a growing pain of overtime, rework and technical debt.

Scope adjustments had been made along the way. But what was now due was the bare minimum the client needed to run their business. There was no fat left to cut out. The system developed was to run our client's core business. And they were very nervous to miss the goal.

Developer with proper protection

On the positive side, (come on, there is always a positive side :) - the team had very committed team members, a weekly dialog with the customer (not pleasant, but still), a positive project manager/product owner and strong backing from management.

When I first met the team, I asked what the biggest problems were.

As a Project manager

- Deliver something with quality
- Team overstressed
- Estimations totally wrong, up to 5 times off for new stuff

As a Team

- Switching too much between tasks
- Doing too many things at the same time

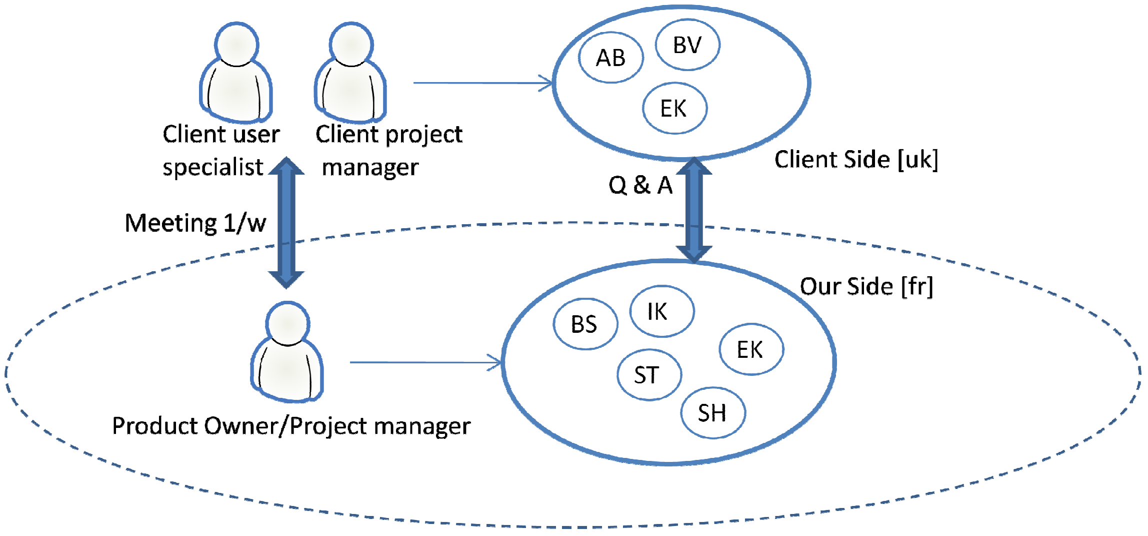
As a Client

- The team is not up to this task

The answers varied greatly based on who you asked. I concluded that there was no help here in figuring out what to fix first.

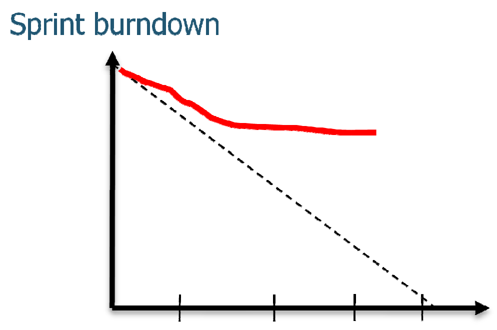
Team setup

In reality there were two teams. One based in Great Britain (client side) and one in France (our side).



The main bulk of development was made in the French team (our side) with some extensions developed by the client. Our product owner and their project manager met once a week to revise prioritization and review demos.

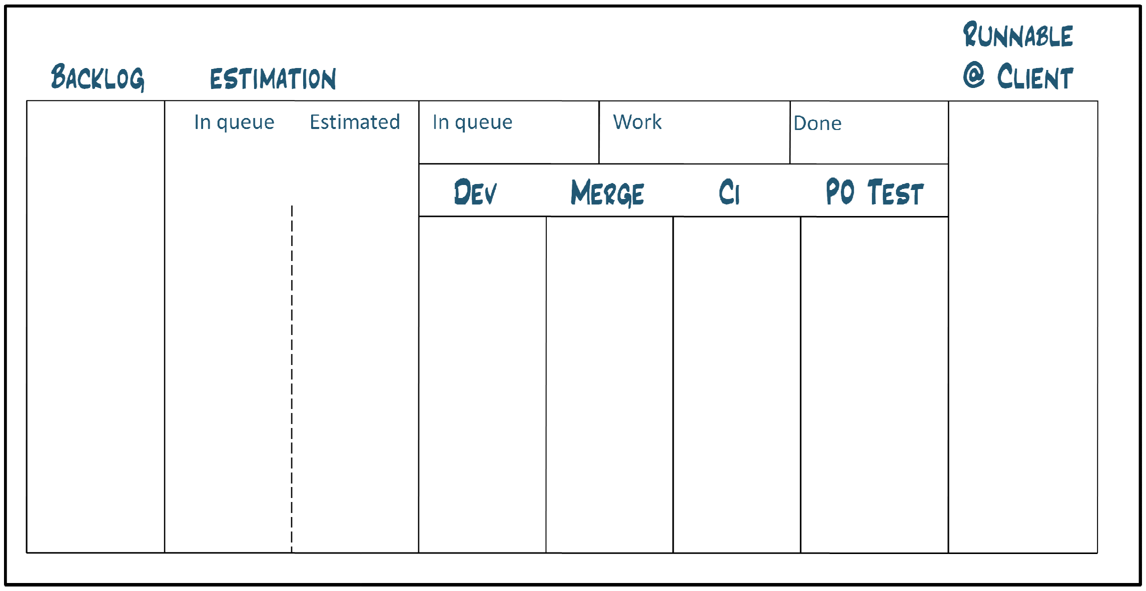
The French team had been using Scrum for a couple of months and the Project Manager was a certified Scrum master. It was not hard to find problem indicators. The team worked massive amounts of overtime, sometimes coding to 1.00am in the morning. The sprint burndown's displayed a worrying pattern:



When asked about how it worked out for them, the developers comment was "We are not sure Scrum is helping".

How kanban got introduced

The team really got introduced to a kanban board by their CEO. He noticed what was going on at another team I was helping out. They had been running kanban for a couple of weeks. He asked if he could introduce a similar board to this team and I said "sure". Next week their first kanban was up and running.

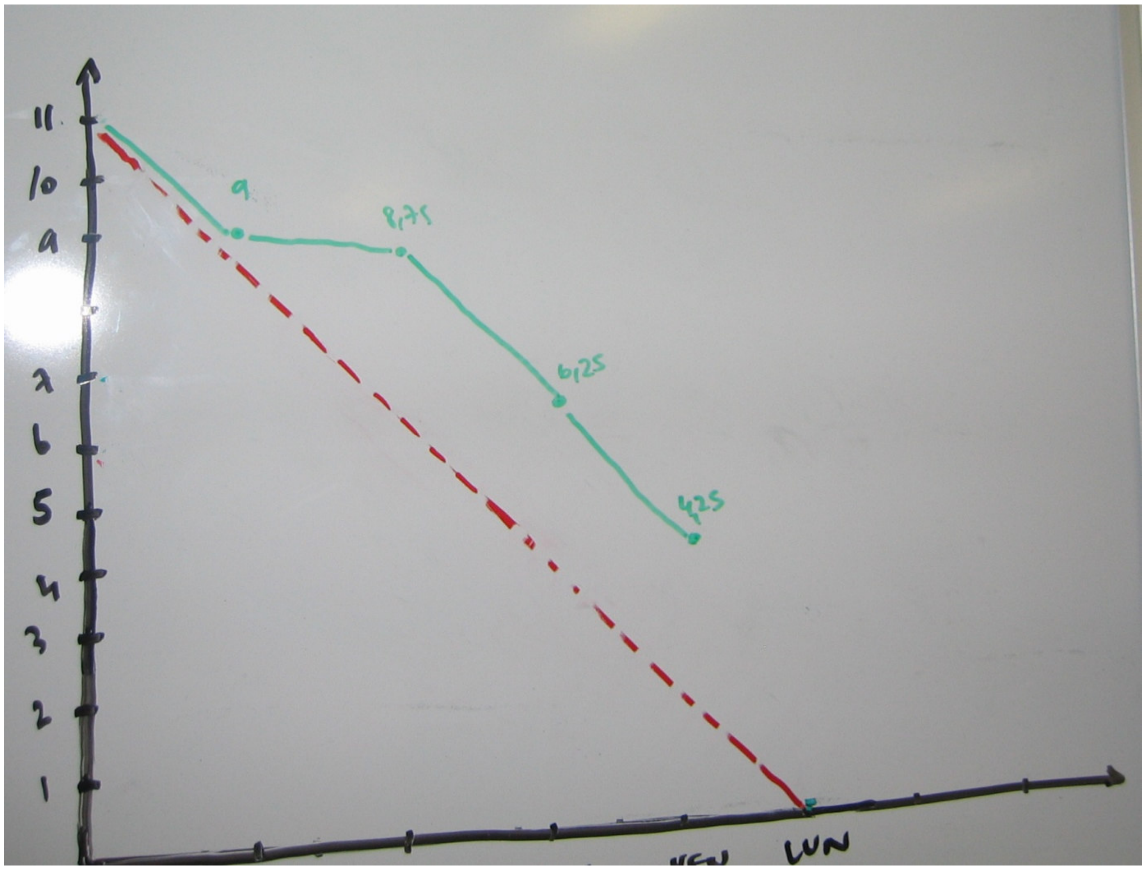


The first kanban board team used. Work in progress was applied to the Dev, Merge, CI and Product owner (PO) test columns. A fast lane existed for urgent items (see top) with the limitation of only allowing one such item at a time.

Nothing else in the existing work procedures was really changed. The only news was the introduction of a work in process limit and increased visibility in the value stream.

After two weeks of Kanban

The teams continued their sprints for two weeks using this kanban board. Below is the sprint burndown after two weeks,



1 Sprint burndown after adding kanban board

It hinted to me the team really could produce working software if conditions were right. About this time I got the opportunity to work full time with the team.

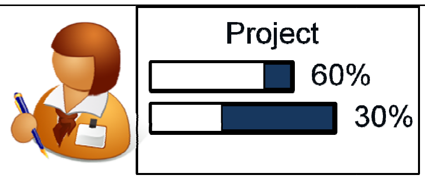
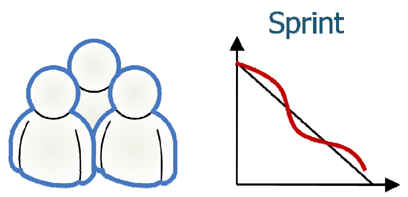
Deciding what to fix first

Two weeks into the process the kanban board displayed some interesting information.

1. Stories got “stuck” in the test phase.
2. Stories would enter the sprint, end up half done then re-enter in later sprints

It turned out that there were several stories proving too hard to complete within a sprint. Unfortunately they were also mandatory to complete the project which meant they could leave the sprint to reenter later, since they did not leave the project scope. This had a demoralizing effect on the team, watching the same problems resurface over and over again, always unsatisfactory solved. The team, often under heavy stress, reverted to the first possible solution in order to make their sprints.

I also noted that the team and project manager was tracking two different things. While the project manager was tracing completion of project tasks, the team was tracking sprints. No common view existed on the project progress.

	
Project manager with task completion focus	Team with sprint focus

Finding time to do necessary improvements

While there were several things we could try out, we had a more urgent problem first: finding time to do things on. The developer's schedules were filled to late evening trying to get the software done. So, we needed to find slack.

Simply estimation

The simplification of estimates started as a way to solve a more urgent problem. I needed some time to talk with the team but they were busy that day, they had promised delivery of story estimates to the client and had the day set off for it. I proposed to the project manager; "if we can finish the estimation faster then expected - can I get free time with the team?" She was happy to try something different so she said "sure". I asked them to have lunch with me and bring the stories along. During the meal, the team passed around the stories and completed the estimation using a simple T-shirt sizing scheme. By the time of coffee, all stories had been estimated and I had my free time with the team.

The estimation scheme we used was very simple:

- Small : 1 day or less
- Medium: 1 week or less
- Large: Anything bigger

Abolishing sprints

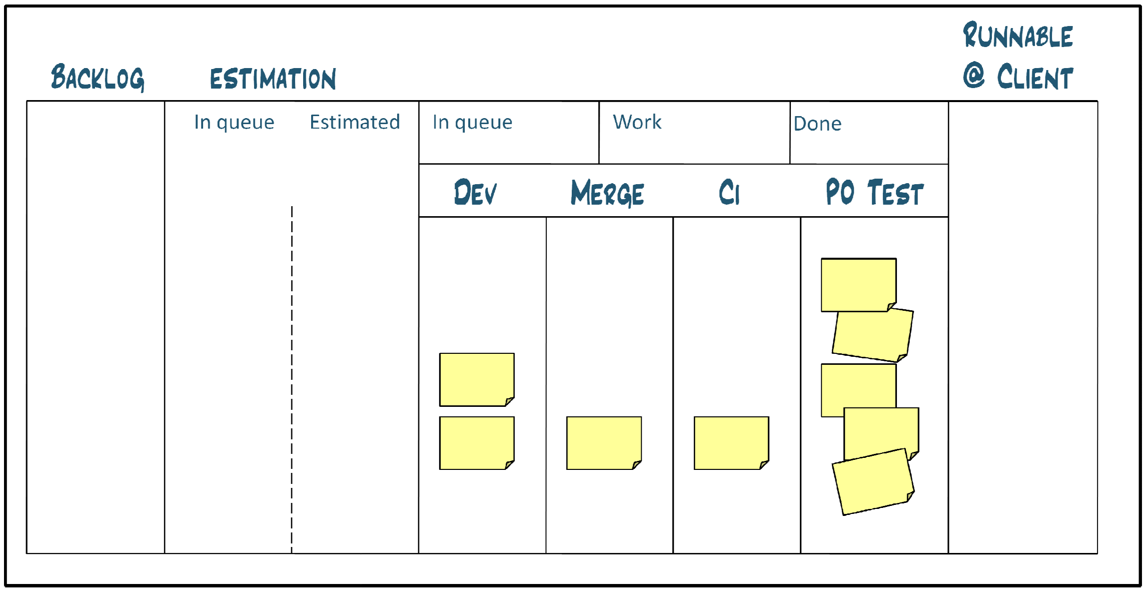
We did not see the value derived from our sprints. For example, they continuously seemed to be broken by outside events. So we decided to try stop sprints, instead focusing on achieving high quality and a continuous flow.

One of the thing we kept was a weekly release cadence. If a piece of work was completed with high quality, it would be passed into the release. If not, they would have to wait until next week. This enabled the project manager to mitigate stories "close to completion" and shift them to the next week's release instead of moving the release for one more day.

The value we got from these actions was less thrashing - stories were now allowed to stay in work until finished.

Addressing quality and development unbalance

It was not rare that the kanban board to display the below pattern:



Stories would aggregate in the test column. We were quick to code but not as fast in testing and correcting. It was our project manager doing the testing so in reality we had an part time tester.

Building in quality

How to fix the unbalance? We started by introducing TDD. The intention was to shift parts of the quality work load onto the developers and introduce less bugs.

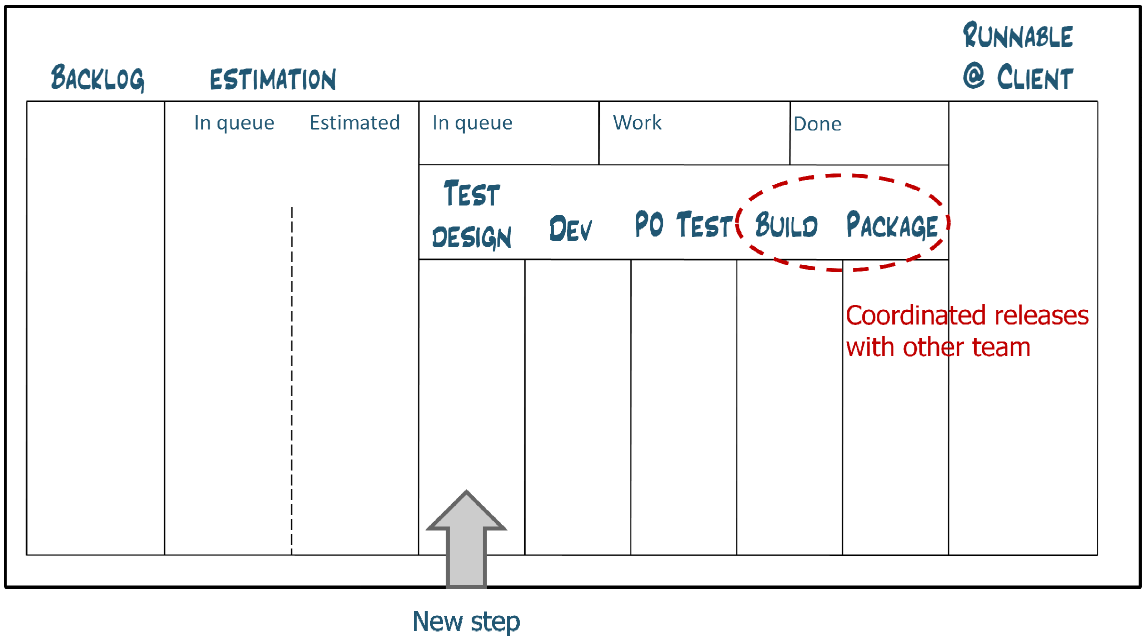
We ran four TDD sessions to teach the team test driven development on the fly. Because of the tight schedule, we had no other choice than to run the training off hours. In agreement with the team, we conducted a set of morning sessions when the team agreed to come in early four days to do TDD workshops.

Right after the TDD workshops, the team addressed a series of refactorings in their code. It turned out the team had known about them for a long time, but now felt they had the necessary tools to deal with them.

How we dealt with test frameworks and technical hurdles

Not all parts of the code could be developed using TDD. For example, there was GUI code were not supporting TDD. Although we had ideas of how to fix this, implementing the necessary infrastructure was time consuming technical hurdle and not something fixed in a day. So we made a compromise:

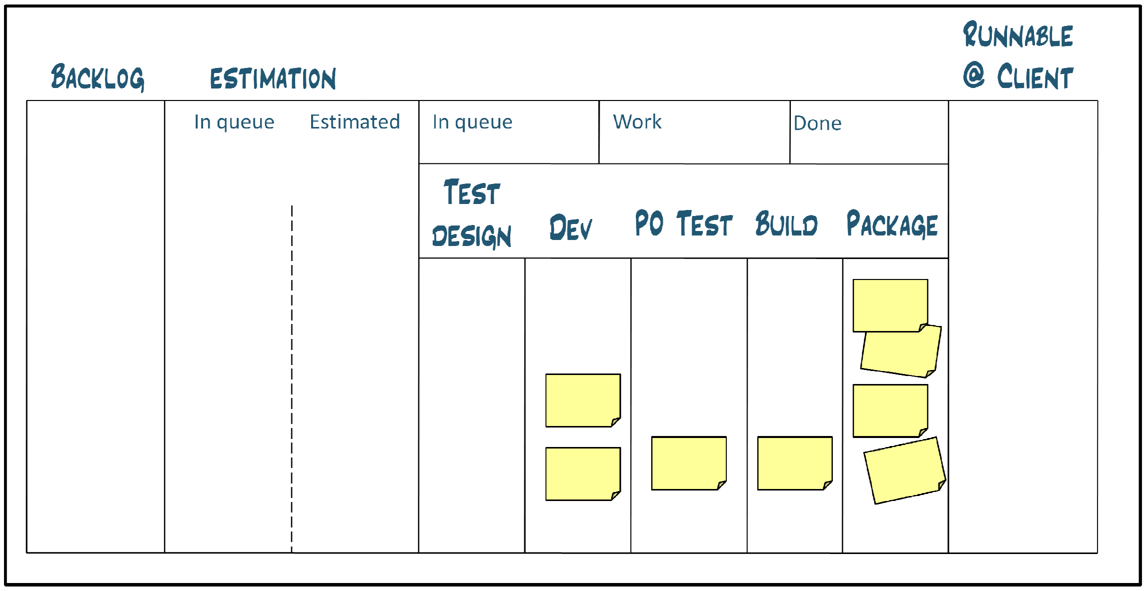
- Design for test were we can
- Do manual testing were we could not write automated test cases



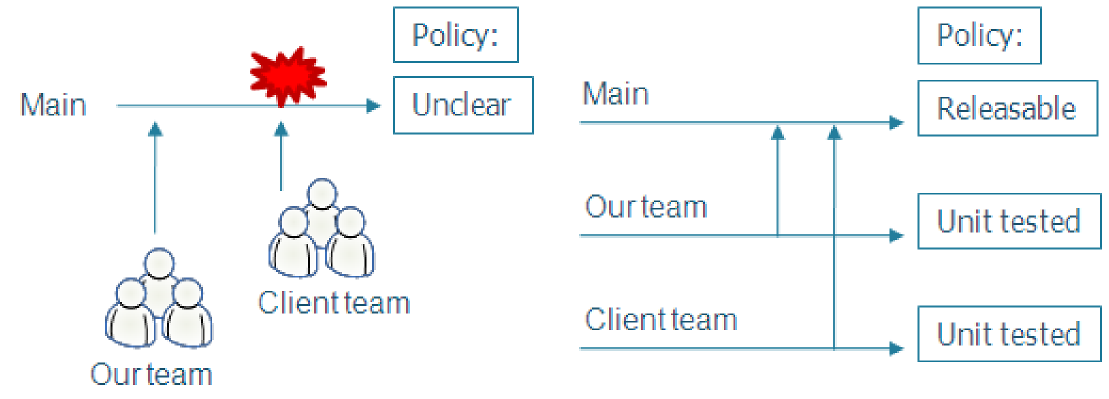
The second step of the kanban board. Test design was introduced as before coding. Team also changed the last steps to build and package, to better support release coordination with the client's team.

How we got to a releasable state

A new problem quickly surfaced:



While the quality of seemed to improve our releases were frequently interrupted by late commits. To mitigate this we decided to change the branch policies.



From **To**

We pushed to get this implemented in both teams; when in place it helped us maintaining a releasable state and mitigate problems while still small.

Small signs of improvements

Even if the pressure still was intense, small signs of improvement surfaced. Comments like "why do we have things like this in our code!" could be heard the and team members started to take individual initiatives in refactoring and fixing quality problems.

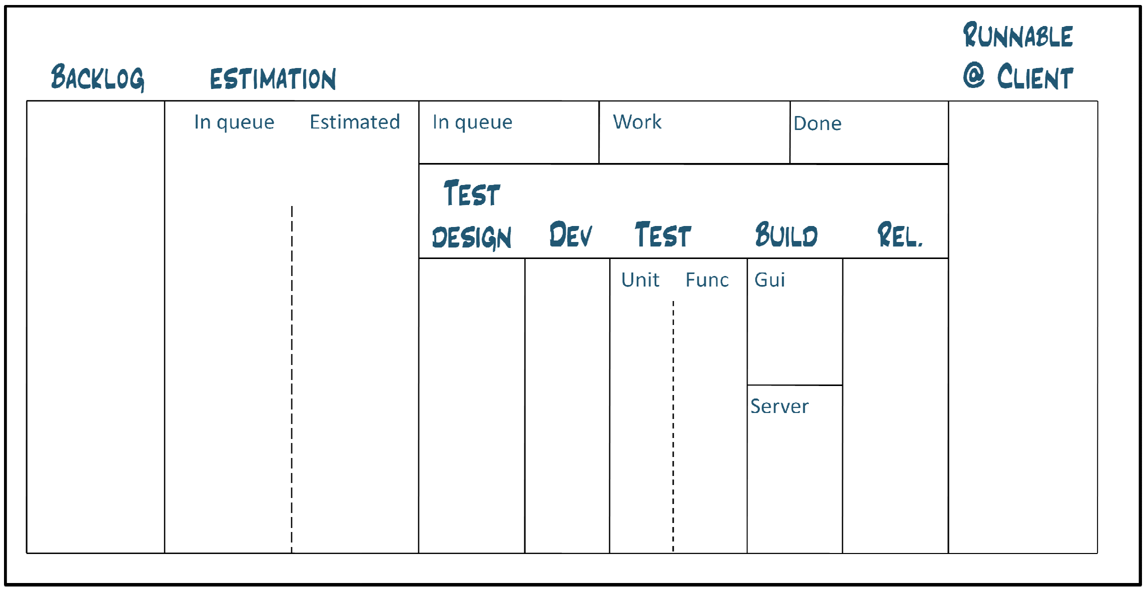
About this time the project manager returned from a client meeting and told me "You know what, even if they are still stressed out, they now trust me when I say we are going to deliver something".

Small indications we were on the right track.

How we got continuous improvement going

So far we had mostly been on the defensive, fighting off problems as they surfaced. Also they had had a support from an outside coach (me). But we needed to strengthen the team's capabilities to act proactively on their own.

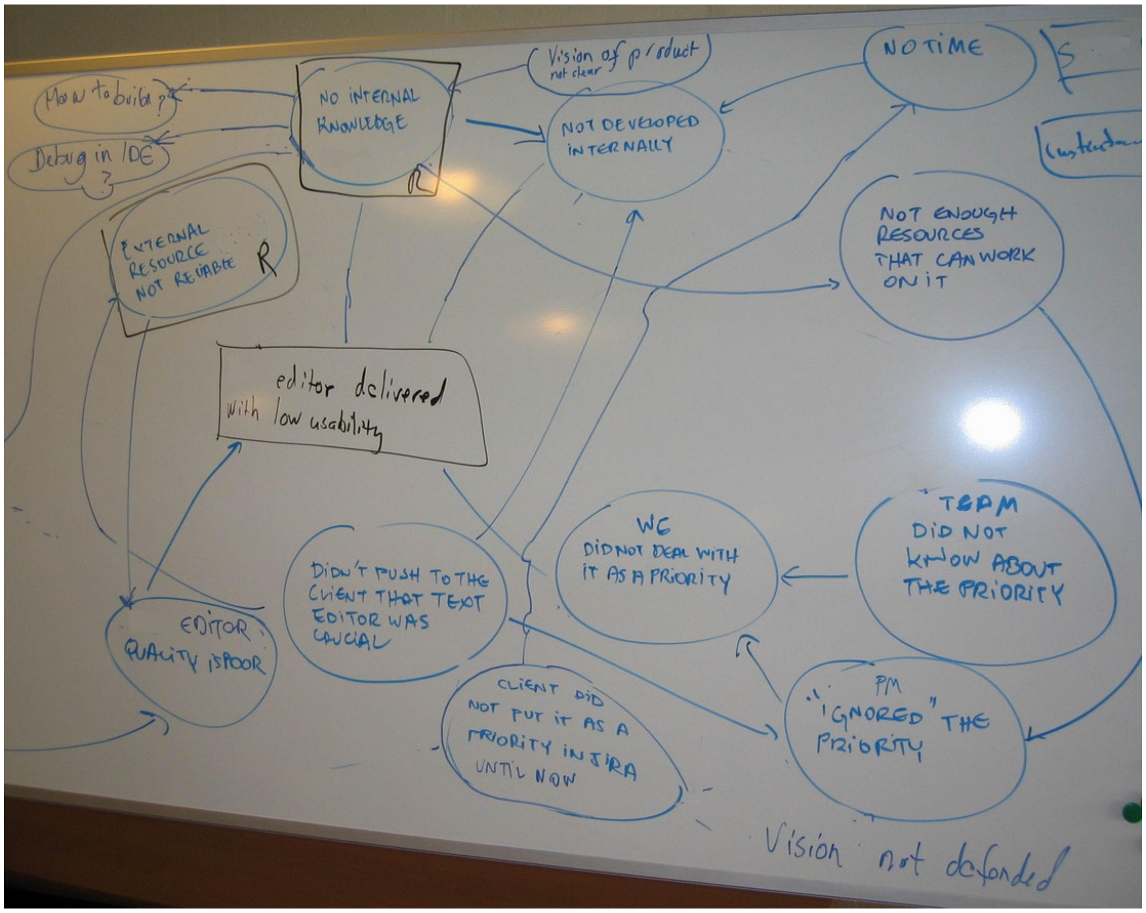
About this time the team starting to take control of the board; adding policies and refining work states.



Updates to the kanban board. A triggering mechanism tells when partial or full builds are needed. Testing is separated into unit test and functional testing to help focus manual testing.

I trained the team in how to discover problems early using the board, in doing root cause analysis and team problem solving. The team started to run weekly continuous improvement sessions together with their project manager. Each session focused on one simple thing: fix one problem every week.

Root cause analysis helped us move from individual problem perception to unified understanding of cause and effect.


How to build?
Debug in IDE ?
NO INTERNAL KNOWLEDGE
R
Vision of product not clear
NOT DEVELOPED INTERNALLY
NO TIME
EXTERNAL RESOURCE NOT RELIABLE
R
NOT ENOUGH RESOURCES THAT CAN WORK ON IT
editor delivered with low usability
WE DID NOT DEAL WITH IT AS A PRIORITY
TEAM DID NOT KNOW ABOUT THE PRIORITY
DIDN'T PUSH TO THE CLIENT THAT TEXT EDITOR WAS CRUCIAL
EDITOR QUALITY IS POOR
CLIENT DID NOT PUT IT AS A PRIORITY IN JIRA UNTIL NOW
PM "IGNORED" THE PRIORITY
Vision not defended

Example of a root causes analysis. A user interface component delivered with bad quality because team was dependent on third party to fix it.

It was fascinating to see how issues started to surface. An example: the problem in the picture above "Editor delivered with low usability" (UI component missing prominent feature). In this case the project manager had known about the problem - but considered it useless to surface since there was no obvious way to fix it. Once surfaced, the problem got addressed immediately by starting to train the team on understanding the code base.

How we addressed cross team problems

A problem that surfaced during the continuous improvement sessions was "how do we get a working cooperation with and transfer know how to the client's team?". To address this, lead developers took initiative to arrange a weekly telephone conference for addressing common problems. We also started a developer exchange program, where a developer from client's team would spend time with us and we would with them to learn their context.

Team members solving problems

How we kept focus the last weeks

During the last couple of weeks the managers and the team worked close together to fix problems that might prevent the release. For example; our CEO ensured availability of a senior developer to speed up fixes in core modules as we got closer to the release.

So.. did we make it?

Yes, the team made the deadline. While this seemed like an unreachable goal two months before, I remember the passing the finish line more like every day work than something extraordinary. The official acknowledgement arrived one week after the release when client gave a green light for the next project.

We all know it is not impossible to meet a tough deadline. Really - all you need to do is work massive amount of overtime. ☺ So - what made me real happy was a comment from the project manager right after the release:

"You know what? The team no longer works overtime."

A look back

I find it hard to identify "one single thing" that made this work. It was more the combinations of many small things accelerating each other

Where kanban really helped was by showing us where the problems were (and *weren't*!). Fixing the problems was up to us.

The good thing was that this information was "cheap". All we needed to do was to put a kanban board on the wall start to use it. No other changes were required. We kept our Jira tracking system, our project structure and way of work. But, as soon a problem was identified, we then turned our full attention to fixing it.

Why didn't Scrum work for us?

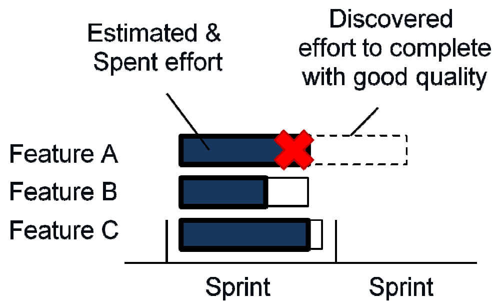
While our implementation of Scrum certainly was less than perfect, I think this also highlights that it *is* hard it is to do a good Scrum implementation. For example, it might even be impossible to enforce a strict "definition of done" if another organization controls when your software is really tested or released.

But, here are a couple of reasons why our implementation didn't work out;

Waste caused by unfinished features

Features were designed to fit into sprints. Once development started developers might discover it' a bigger chunk than they could chew off thus de-scoping them late in the sprint. In the next sprint, another feature with a higher business value is moved in. Repeat this and the project plan starts slipping. When we moved to kanban features were allowed to stay in work until finished and when finished they would be included in the upcoming current release.

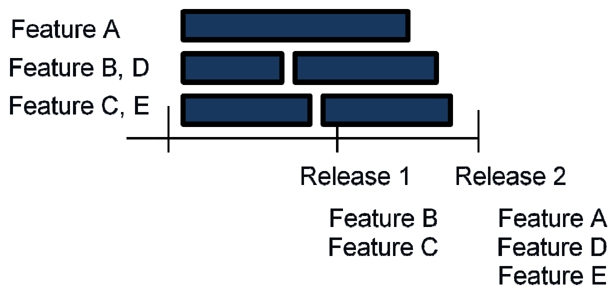
Before (Scrum)



Feature A is planned into the sprint but discovered larger than original scope. At the end of the sprint it is de-scoped to complete the delivery. In the next sprint another feature is moved in with

higher priority from the backlog. In this way time spent on Feature A is either turned to waste or it had to be released prematurely with low quality.

After (Kanban)



Features stay in work as long as required to complete with good quality. As soon as finished it is moved into the upcoming release which were made on the same intervals as previous sprints.

Key refactorings were held back

Prioritizing features by business value combined with mounting pressure from client to get back on track with the project meant important refactorings were held back. And if the team managed to convince project management to give a refactoring a go, they were not able to finish them within a single sprint. In this way client's project management started to treat refactorings as pouring money down the drain.

Solving problems required a cooperation from multiple stakeholders

On one occasion the team wanted to refactor a key component outsourced to an external vendor. Pulling this through required cooperation of our team, platform specialists, management and the client. Thus, even if the problem might have surfaced by the team - it would not get solved until multiple stakeholders put aside effort to solve it. To solve problems like this it was important to involve multiple stakeholders and isolating the team into sprints did not help in that process.

Estimation consumed precious resources

.. Without adding additional precision.

And let's face it – it is not all about Scrum. There are multiple dimensions that need to pull together to make a software project work.

What about other solutions?

There always exists more than one solution to any problem! So let's give this some thought.

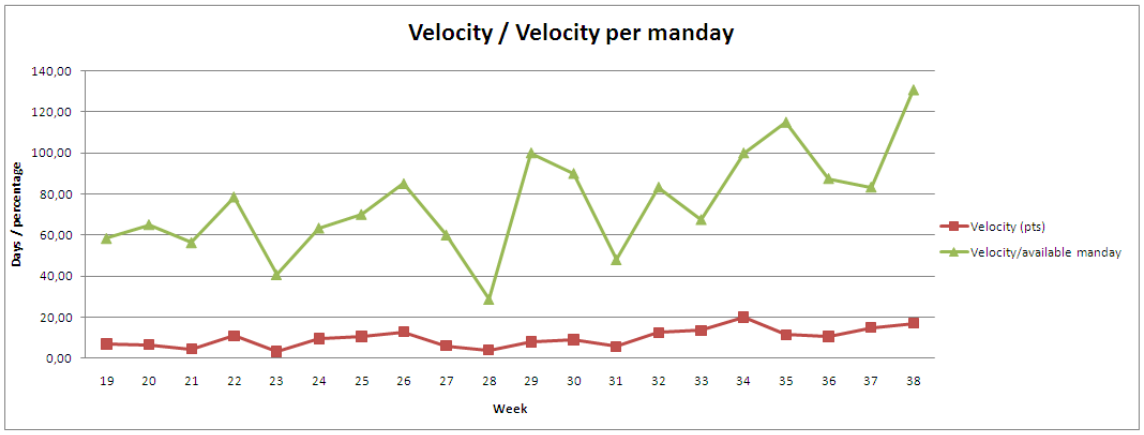
"So it looks like you were thrashing, why not work in longer sprints?"

That is also an option. Our problem was we had not enough trust selling this to the client. I am not sure a message like "we are going to work in *longer* sprints and keep away while we do" would have produced a positive reaction.

"And what about your definition of done?"

It is true we were working in a "scrum butt" way. But - we were not the only party in the project. We did not control the client's deployment procedures for example. Changing the definition of done required having the client with us, accepting the consequences. In reality we did something similar by providing a longer visibility downstream into the value chain.

What about data?



Velocity / Throughput

The average throughput in May was 7.25 story points per week and in Sept 13.50 sp/week. (red line)

Velocity / Man-day

If we look at the throughput per man-day, the May average was 0,64 and in Sept 1,04 (green line - vertical axis in graph scaled in percentage) .

Where are we today?

The team has pushed on and implemented most of the testing frameworks they wanted to put in. The reason I mention this is to show that you can overcome had technical hurdles and difficult project management sells. In the beginning we could not envision the moment we would get the necessary time to put these fixes into place - but now they are there.

Much of the skills that this team learned have been transferred to the client's development team. The challenges today are related to transferring the knowledge to the client's IT organization and project management.

Views of a team member

> "The hardest part is to train yourself to embrace a new mentality. To realize that if a task is coded, this does not mean the task is completed.
>
> Kanban 'forced' us to think about quality. When you have a column called 'function test' on the board it's a little bit hard to ignore it :) It forced us to stop the coding, coding, coding chain.
>
> I think, the best lesson we learned was *always think about quality*".
>
> - Mariana, developer

Kanban going wild!

Late into the project a girl from H&R approached me and asked if I had any Ideas that could help decrease the stress she experienced. I was skeptical at first but figured out that frequent reprioritization of things she had started to work on was one of the culprits. We discussed some options then I drew a simple Kanban on the glass wall of her cubicle. I asked her to tell her instructed stakeholders to put their requests in priority in the "in queue" - but leave any started work alone.

One week later I passed by her office and noted there were Kanban systems all over the walls. A kanban system was had been started by the all the staff around her – economy, marketing and IT.

Kanban threatens to take over the world..

I hope there is something here to learn, I learned a lot.

Mattias Skarin
Stockholm, April 2010