Maintenance using Kanban

Beijing experience

Rune Hvalsøe

Executive summary

- We found that switching from traditional way of working to Kanban gave us:
  - 70% performance increase
  - Better overview of remaining work
  - Better priority – Always fixing the most critical issues first
  - Better knowledge sharing
  - Happy developers
  - Less stress

Story summary

- This story is taking place in Beijing November 2008.
- I was responsible for building up UI development and maintenance on Nokia's S30 platform (low cost phones).
- We were struggling with the lack of skilled SW developers and big maintenance load from various projects.
- The developers was working over time when they were getting close to a project deadline, to keep up with the number of issues.
- The issues was assigned to the developers, typically the developers had 10-30 issues assigned and was working on 4-6 at the same time, to be more effective.
- This had been the way of working for many years and the experience was that the developers was burnout.
- The department was able to handle the maintenance, but most of the time we did not have the extra resources to do any new development.
- We had started to use Scrum for new development in October 2008 with fantastic results, so we wanted to try the same concept with our maintenance, we were measuring the average number of issues that we fixed per week, so we was able to see if our new way of working was making a difference or not.

Story summary

- We quickly realized that Maintenance and Scrum did not work well, however after adjusting the model a couple of times, we ended up with a model that was looking very much like Kanban (another Agile discipline) – described in details in the coming slides – the main idea was the each developer was only allowed to work with one issue at a time.
- The developers was no longer working over time and actually had a little extra time during their working hour (i.e. waiting for compilers etc.), we were however able to handle the maintenance load and even had extra time to start new development projects.
- Our first thought was that the number of issues had decreased and this would be the reason why we had the extra time, however when we looked at the number of issues that we fixed per week, we realized that we had increased the productivity with 70%!
- The reaction from the higher level management in Beijing was mistrust, they believed that we were delivering poor quality and that was the reason for the big increase – so we started to investigate if we had lower quality in the new way of working, we looked at the code and at the amount of "bounce back errors", however the quality was still very high and on pair or better compared to our "old" way of working.

Story summary

- The consequence was that the section could start new development even in the phase when projects was about to close, we even had resources to help other sections when they had extra workload (they did not use Agile principles and was still working OT).
- We started to rotate people – the rotation secured that everyone did maintenance from time to time and new development from time to time
- We continued to work with this model, and 2 years after I left, I talked with the manager for the section about how things was going, and he told me that they was still measuring the average number of issues fixed per week, and from time to time, he saw that the number was dropping, he talked with the developers and found that a few had picked 3-4 issues to be more effective. After getting everyone to return the issues and only working on one, the productivity went up to the normal level.
- Imagine that we start to use this way of working, i.e. get management and development resources to understand the importance of not doing task switching in the daily work, imagine where we could use the extra time that we would gain…

Original idea

- Our original idea when we started the “new way of working” project was to use the findings from our initial test with Scrum
  - Very strong commitment from the team (we saw that Scrum created a very strong commitment from the team members)
  - Better quality using DoD (Definition of Done)
  - Better overview of remaing work
  - Better time estimates, i.e. How much work is left and when are we done
  - Flexible use of resources, i.e. Focus on fixing most important issues first, meaning that you should not only work in your normal domain => Spreading knowledge

Conclusion

- Our findings was not even close to our expectations
  - Trying something ”new” sometimes give the **un-expected**
  - 70% increase in output, this was clearly not what we expected
    - We removed an unidentified time consumer – task switching, people (developers, project managers and people managers) did not see that this was causing delays in deliveries, actually most people felt that they were less effective after the new wow, because they had idle time – however statistic showed that we were 70% more effective due to the focused way of working – I guess at this time, most people was admiring other people who was solving many complex things concurrently, an interesting article about the topic can be found here: http://blog.codinghorror.com/the-multi-tasking-myth/
  - Better overview – we clearly got a much better overview with our great wall, sub-groups and priority system

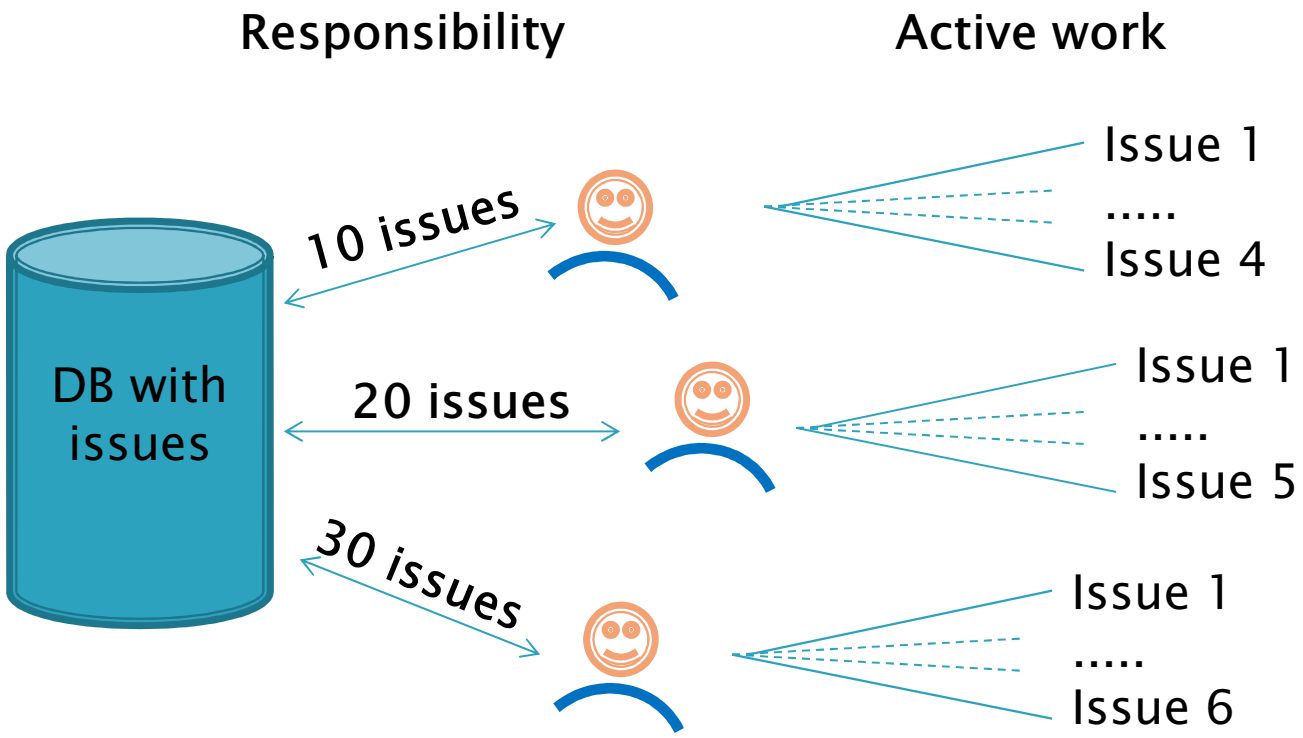
Conclusion

- Kanban – We had just learned about Scrum and tried it with great success in a small team (we did not completly follow the Scrum rules, i.e. I believe in adobting and testing rather than following rules), we found that Scrum does not work with maintenance for several reasons, I changed our way of working and we ended up with something that looks very much like Kanban (which I learned about later)
- Knowledge sharing and cooperation, we forced the developers to fix high priority issues first – no matter which sub system they use to work on – when someone selected an issue outside their normal competence area, they would request (and get) help from the ”experts”, the result was a much better understanding of the system, and a much more flexible group of developers, i.e. they could work in several areas and had a better understanding of how the different systems work together.
- Less stress and better quality – The developers had more time to reflect while waiting for the compiler etc.

Details

- The following slides describe the original setup and the new way of working in more details

Original way of working



Original flow

- Developers was assigned issues directly, often they were responsible for 10-30 issues
- Developers was often working several issues at the same time, upto 4-6 issues, we often saw that highly skilled developer was working on more issues

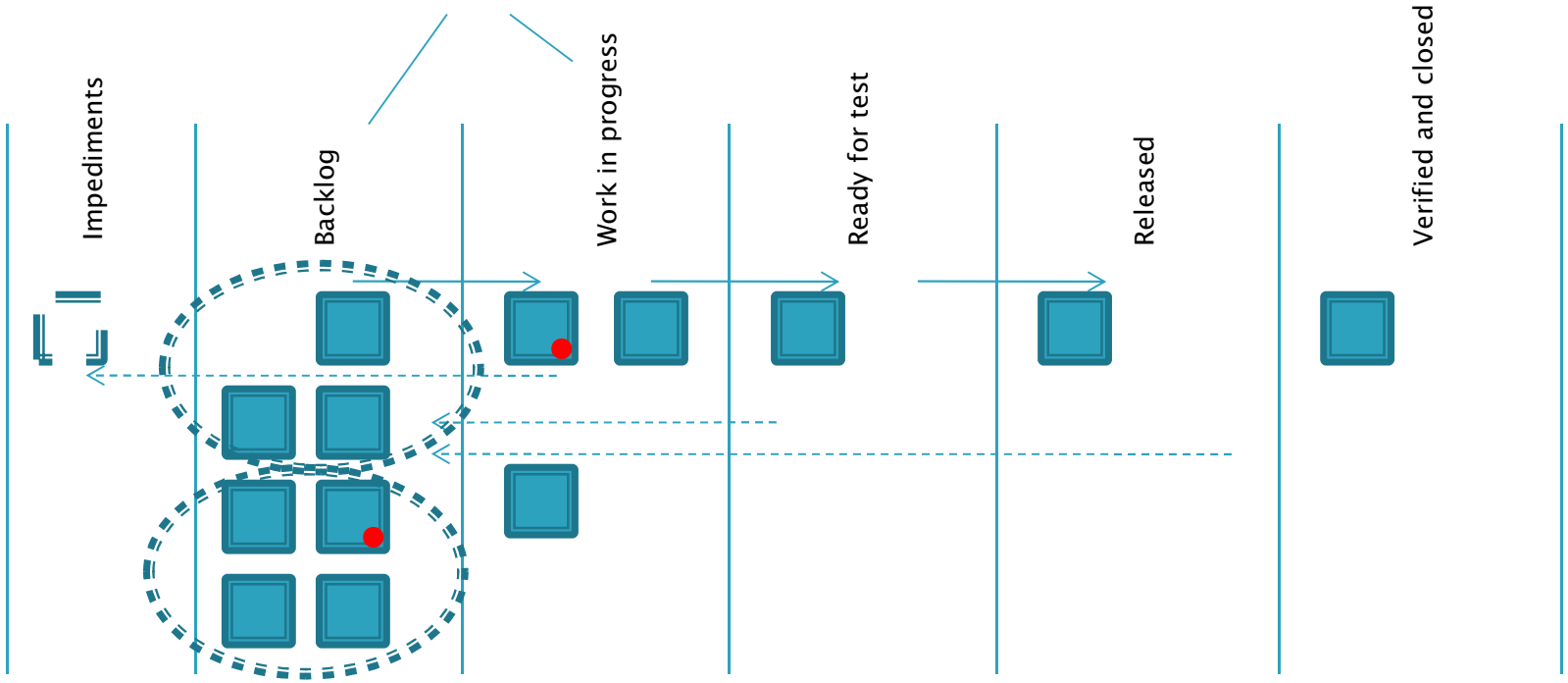
Kanban flow

Errors

Kanban Chief

Kanban chief will put errors on the prioritized backlog when they have sufficient description including needed log files with relevant trace

Developers can only have one active error(s) at a time in work in progress.



If an error require extra trace etc, it will be moved to impediments and developer select a new error while waiting for input – when the input comes, the error manager will move the error back to the backlog. Show stopper errors would be marked with a red dot to indicate that they have to be fixed first.

Kanban flow

- Issue arrive to “Kanban chief’s” inbox
  - Kanban chief investigate that the issue has a good description, enough log-files etc. – i.e. that the issue is ready for the developer
  - “Kanban chief” will prioritize the issue and put it on the teams wall.
  - Issues are grouped when put on the wall
  - Show stopper issues (SS=critical) got a red dot
  - The “Kanban chief” will be responsible for the issue until a developer have selected it.
  - We call the “Kanban chief” for the QA-function at Sony.
- The developer will select “*one*” issue from the wall of issues, if any issue is marked as a SS, then that issue have to be selected no matter the developers competence, but those who have the competence will have to help this person if needed, this also ensure that there is a knowledge sharing in the section – the developer will put himself as responsible for the issue.

Kanban flow

- If a developer need additional information, the developer will request the additional information, write all known information about the issue in the error report, so others (or himself) could get faster up-to speed, put the issue back as “impediment” and assign the “Kanban chief” as responsible – the developer then select a new issue from the wall, like normal.
- When the “Kanban chief” receive the requested information for issues marked as impediment, he will move the issue back to the “Backlog” and it will be possible for a new developer to select it.

Kanban flow

- When the developer have fixed the issue, he will test it and call for a review – when this is done, the issue will be send to pool for testing, in Beijing we had a special team sitting next to us who did extensive testing on many different variants.
- When the issue has been delivered for test, the developer will select a new issue.

Team of 12 developers working with maintenance – switching to Kanban