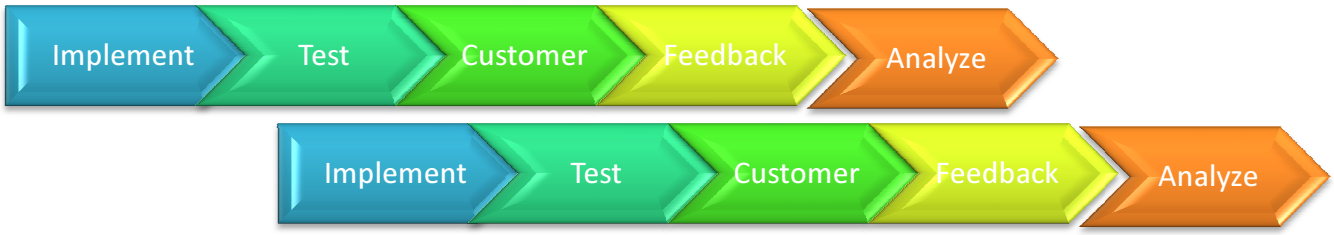
I did a presentation in Malmö May 31 with the title “Closing the loop”, this document describe most of what I said at the presentation.

What are we trying to achieve?

We want to deliver the best possible user experience to the end customer and give them the latest and greatest.

How does it work before we started to change our way of working?



We developed new software over a number of months, the phone projects worked according to water fall models and wanted us to deliver our software long before the phone was released to the end customer, so that the products could be tested before final release; after the phone reached the end customer, we collected information from various sources to get feedback from the end users about our implementation.

The challenge we saw was

1. The software we delivered, often had a couple of usability problems
2. The time from we finish our implementation to the end customer receive the result is at least 2-4 months
3. The time from we start our implementation of a feature until we get feedback-data to analyze and use it to improve our next implementation is very long 12+ months
4. The feedback we get from end customers is received from various sources and hard to collect and often not very precise, only the most enthusiastic (both negative and positive) write on blogs
5. We rarely had a chance to change an implementation on an end customers phone after they bought it.

Before I talk about how we deal with the challenges above, I will shortly describe our teams and how they work.

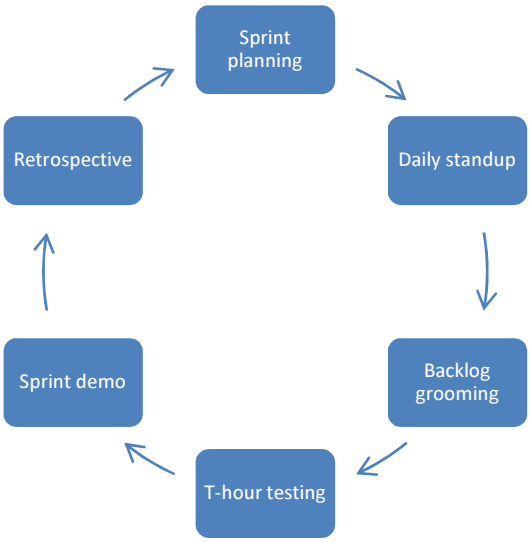
We work with static teams, i.e. we try not to change the members in the teams, but keep the same people in the team, this gives the best options to build self organizing teams according to the FIRO model.

Each team have

- 6-8 software developers, we do not have any testers allocated to the team.
- a FTL (Feature Team Lead) – a Scrum coach and the teams interface to the projects.

- a PO (Product Owner) with an allocation of approximately 20%, this means that the team (developers and designers) is much more involved in creating the backlog and doing the priority
- an interaction designer to help the team to create the blue print of the feature
- a graphical designer to create the graphic, review the implementation

We work closely to the Scrum description in most books, but as with everything else, we twist it to suite our way of working.



Most of our teams work together with other teams within Sony (they do not work with Agile), and we need to request translation of text 4-6 weeks before we need them.

The team is sitting physically together with the graphical designer, the PO try to participate in as many team meetings as possible, the FTL is sitting close to the team.

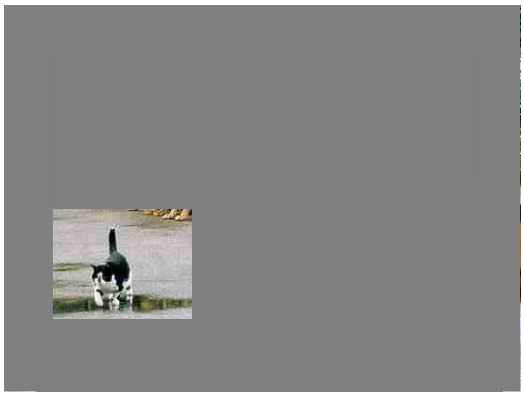
We have the traditional sprint planning for the coming sprint, we have daily stand-up where the team report and align current status, we try to have a backlog grooming in each sprint, where the team go through new stories, discuss, get a common understanding and estimate the stories. The team has at least one T-hour in each sprint, the T-hour is where the team test the latest SW, finally the team end the sprint with a demo and a retrospect.

We put a lot of effort in securing the quality before we deliver, it is the teams responsibility that the quality is good and they are also the once who test the software, however we do have an organization in China who help us to test our software after we have done the delivery, i.e. the make sure that we don't degrade our quality over time.

The first challenge I listed was about releasing SW with usability issues, even though we have a high technical quality and follow the blueprint description closely, we don't always find all usability issues.

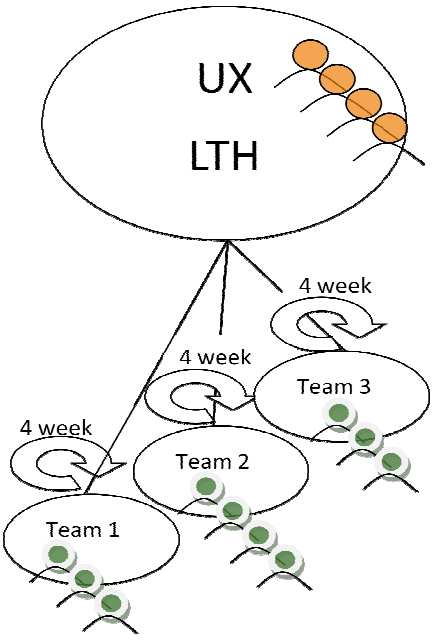
The idea we got started when we invited a group of employees from HR (Human Resource) to participate in our test, instead of the traditional T-hour that we did, we had a session of 1½ hour where the testers from HR was asked to solve a couple of user scenarios while the developers was watching, we worked in pair, i.e. one developer allocated with one tester. The result was an eye-opener for the team (developers, PO and the interaction designer), the "new" testers had problems understanding the intended flow, this was new to us, as our normal T-hour testing did not identify this, the T-hour is done by the developers and interaction designers, but they know the phone and the feature so well, that they don't look for problems in the flow.

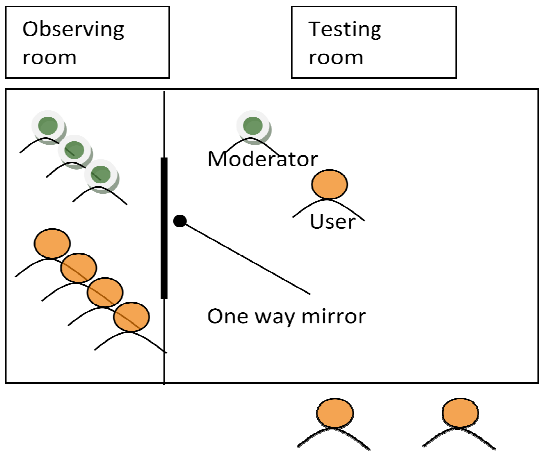
The problem is described by Daniel Kahneman in his book “thinking fast and slow” as WYSIATI (WhatYouSeeIsAllThereIs), our brain are lazy and we don’t challenge ourselves – there is a good example on the Internet where a group of people are asked to count number of times a ball is passed between basket players, most of the participants does not notice that a “monkey” jump into the picture… i.e. in the picture below (which I borrowed from the web) we see the cat, I would claim that we are only looking at the cat in our testing and forget to look for other challenges:



The testing with HR was what made us look for other ways to test our software, we realized that we need to involve people outside our team in the testing to help us to see the things that we cannot see.

We got in contact with LTH, the idea was to use the students who had been having courses in usability, and at the same time give them a chance to see how it works in the “real” world (we probably all know the feeling of reading about something and then trying it out in practice), the students who have had courses in usability are also better at explaining the problem that they face. We started a pilot project in 2012 and we are still using the same setup (we have done a few minor changes):



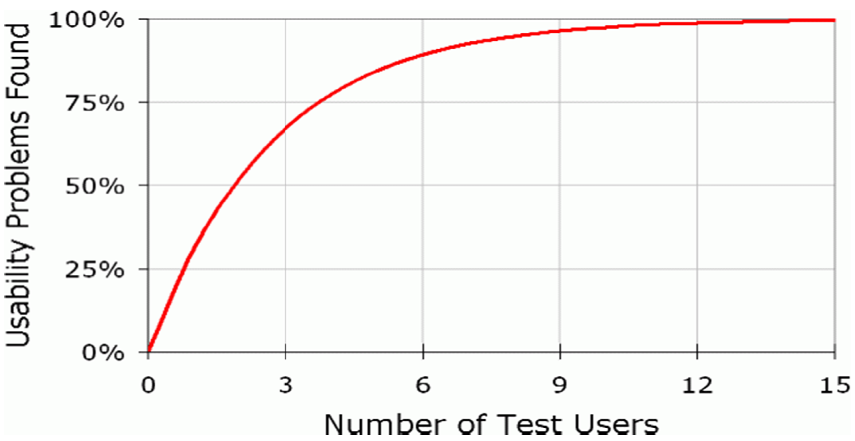


The team plan for a usability study after every 2. Sprint (4 weeks, give us enough time to implement a sufficient number of stories), the test is time-boxed to maximum 3 hours and involves 3 testers:

- 0-5 min: we explain the concept to the students
- 5-35 min: first test, the tester is given a number of tasks to solve in our application, the tester is guided by a professional moderator, the team (developers, PO, interaction designer) is sitting in a room next to the tester with a one-way mirror, the 2 remaining students is sitting in another room, where they play around with one of our phones to learn to know it - we learned that far from all students know to use a Sony Android phone, so if a student have been using a Sony phone before, this person is first to do the test. The students also have to sign NDA to prevent confidential information to leak.
- 35-65 min: 2. tester is doing the test, the first tester is now sitting together with the team (to give them a chance to see how this works in real life)
- 65-95 min: 3. tester is doing the test, the first tester goes back to the extra room and the 2. tester is now sitting together with the team (we learned that it is not a good idea to have 2 students in the observation room – less focus)
- 95-155 min: we meet in the extra room, developers, PO, interaction designer, moderator and the 3 students. We discuss and list all the issues that we have seen, the students are very engaged in this discussion
- 155-165 min: the developers estimate the size of the issues (Small, Medium and Large) to give PO and interaction designer an idea of how complex it will be to "fix" it.
- 165-170 min: everyone is voting – 3 votes – for what they consider most important, we do blind voting with 2 color post-it, one color for LTH students and another color for Sony employee, people can put votes on same issue or several issues
- 170-180 min: PO and team select 1-2 issues (important not to select all, if you select 1-2 you will implement it, but if you select everything you will most likely not get any done) which they will implement before the next test

We use new students each time we test to prevent that they get to know the features too well.

We only use 3 students, but we believe that we find the most important usability issues, Jacob Nielsen (a UX guru) have done some investigation in this area, and on his web-site you can find a description of his findings, the graph below is from his website http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/ - the recommendation is to use 5 testers, however we have decided to only use 3 testers due to time limitations but we perform the test frequently (every 4 week).



Source: Jacob Nielsen

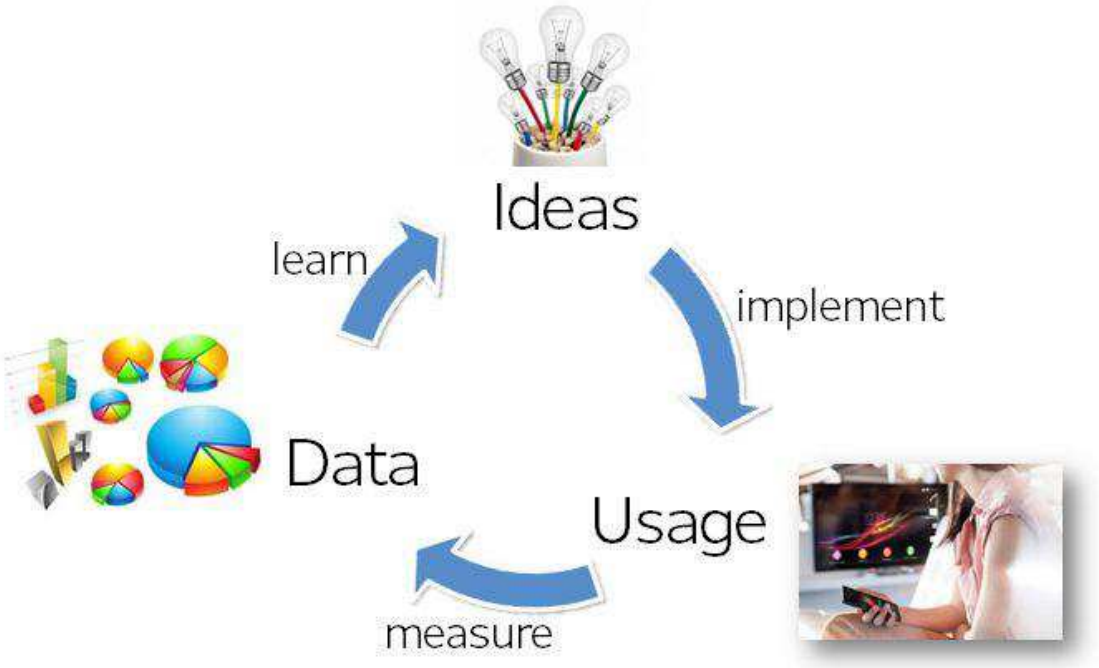
The benefit from the usability testing with LTH is not only to identify the challenges that the students help us to find, but it is also about giving the developers, the interaction designer and the PO a much wider view on the end customers behavior and knowledge. Even though the interaction designer create a good blueprint, the developers always interpret it and need all the knowledge to interpret with the end user in mind and also be able to stand up and challenge the design in the blue print.

We have seen a positive development in the usability after we started to do the frequent usability test, and the team is very positive to the input they receive from the test.

The remaining challenges I listed (2-5) is about being able to receive data from the end customer and be able to update the end customer's phone frequently, to deal with this challenge, we have done a couple of changes in our way of working:

We have implemented Google Analytics (GA) in some of our features, to collect anonymous data from the end user – we have a setup where we only collect data when the user is online and we only collect data from a very limited number of users - less than 10% selected randomly. This kind of data collection has been used by many others and is used every day when we browse the Internet on our computers, however this is the first time that we use this statistic.

We have started to use Google Play and an internal developed tool to send out software updates to our end customers, we have been sending out FOTA packages (full phone software package) before, but this is the first time that we use it to send out updates to our applications.



Our ambition is to have a 6 week cadence, i.e. from we send out a "new" feature, we want to be able to analyze the GA data from the end user and improve the feature – if needed.

We also want to be able to release new features to the end customer every 6 week (or when we have enough to bring value to the end customer), so that they always have the latest and greatest when they buy a Sony product, kind of the same thoughts when gas-stations have loyalty programs for their customer, i.e. people who buy a Sony product shall know that we do what we can to always give them the latest and greatest.

Another improvement that we have done – which help us to reach our target of 6 week cadence – is that we have been doing "4+4 planning" for a fairly long time now.

A challenge when we work with Scrum in 2 week cycles is that the team only know what will happen the coming 2 weeks, and in the middle of the sprint, they only have an overview of what happens the coming week – we try to avoid waste in Scrum, but planning a few sprints ahead is normally not waste, we plan 4 sprint ahead where we do our best to make it realistic, and additional 4 sprints to have an idea of what will happen afterwards (but only rough planning).

The purpose of the planning is to give the team an overview of what will happen the coming 4 sprints, the team identify dependencies in the coming sprints, so that they don't suddenly realize that they needed to do something a few weeks before when they start a sprint, i.e. our translations take 4-6 weeks, we may identify dependencies to other team etc. – these are identified and the team deal with them just in time to avoid problems. It turns out that identifying dependencies like this is preventing the frustration that otherwise hit a team when they realize that they cannot implement the sprint, simply because they should have started a minor activity a couple of weeks before.

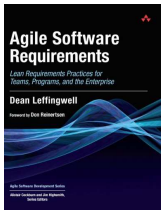
We have had times where we needed to change a coming sprint, move stories around, but it turned out that it was easier with the 4+4 planning available, as the output from the planning is 8 flip-over (one for each sprint) with all stories on, and re-arrange stories is fairly easy when you have them all available like this. Another output from the planning is a short objective for each sprint, and it will be very visible what the team will be able to implement at a given time.

4+4 planning engage the team in a new way, they become responsible for the implementation and get a completely different commitment; all teams who try out 4+4 planning has been very skeptical at the beginning, but they usually end up promoting it to colleagues when they have tried it 1-2 times, the benefit is much higher than the cost.

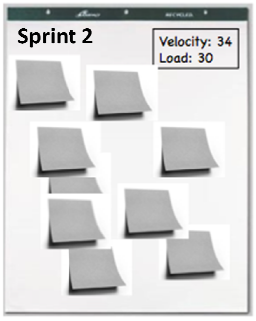
We repeat the planning every 4 sprint (8 weeks), usually we do the 4+4 planning at the end of the 4.th sprint, and we then break down the first sprint in our 4+4 planning to tasks, however we don't break down the other sprints into tasks before the traditional sprint planning, however selecting the stories and doing the normal sprint planning is much faster, as the sprint is roughly planned.

The outcome from our 4+4 planning is shown in the figure below, it is important to remember to have a "mature" backlog before you go into the 4+4 planning, i.e. the stories need to have story points, it will be impossible to do the planning otherwise.

The 4+4 planning take up-to 1 day to execute, it takes less time if it is only one team and they have done it before. A complete description of how we do 4+4 planning is a description in itself, but it is inspired by Dean Leffinwell's "Agile Software Requirement" – a few pictures from one of our 4+4 planning is also attached:

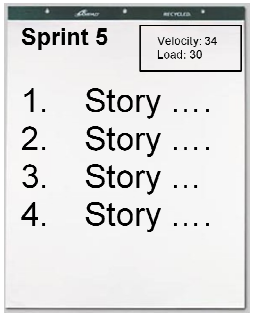


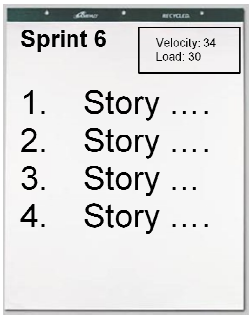

Sprint 1
Velocity: 34
Load: 30

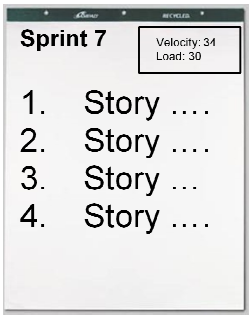

Sprint 2
Velocity: 34
Load: 30

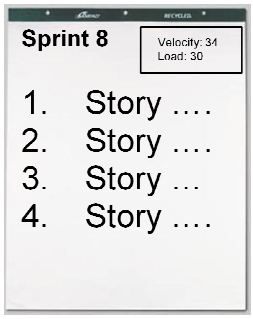

Sprint 3
Velocity: 34
Load: 30


Sprint 4
Velocity: 34
Load: 30

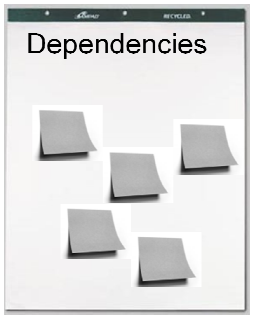

Sprint 5
Velocity: 34
Load: 30
1. Story ….
2. Story ….
3. Story …
4. Story ….


Sprint 6
Velocity: 34
Load: 30
1. Story ….
2. Story ….
3. Story …
4. Story ….


Sprint 7
Velocity: 34
Load: 30
1. Story ….
2. Story ….
3. Story …
4. Story ….


Sprint 8
Velocity: 34
Load: 30
1. Story ….
2. Story ….
3. Story …
4. Story ….


Objectives
1. ….
2. ….
3. …
4. ….


Dependencies


Risks