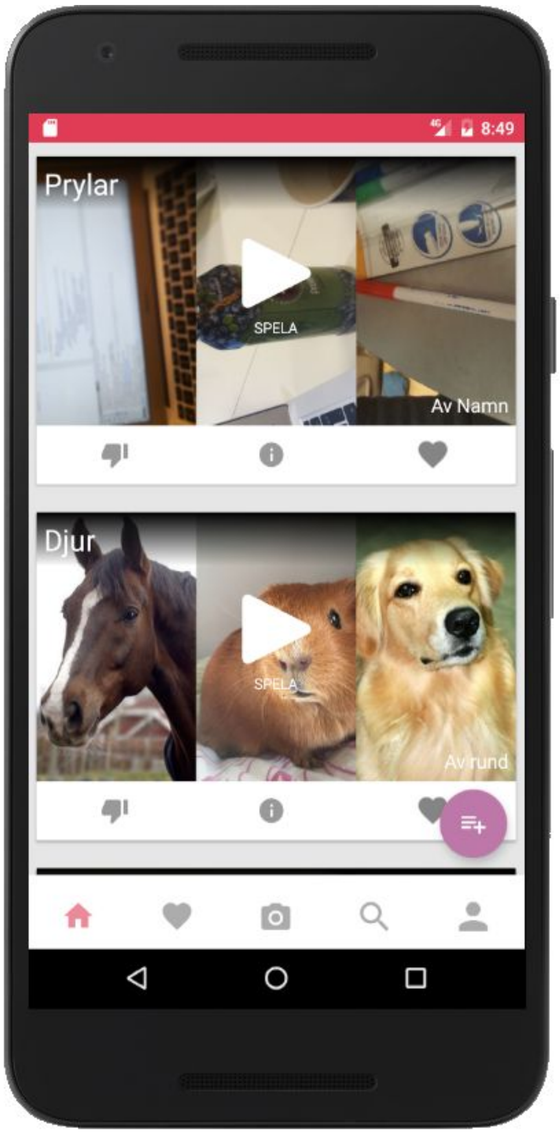
Reflection report - Kasslr



GruppN

Adam Andreasson
Arvid Björklund
Andréas Erlandsson
Daniel Illipe
Frej Karlsson
Marcus Randevik
Patrik Olson

1 Introduction

During the last ten years, the world has come to face one of the biggest humanitarian crises in the history of mankind. UNCHR reports that a staggering 65 million people have currently been displaced from their homes and 21 million of them are refugees trying to seek asylum in a new country. This is more than any point in the history of mankind and has lead to a global topic at the top of United Nations priority list.

1.1 Background

Though Sweden has been one of the most generous countries when granting asylum to refugees, the work of integrating new people to the swedish society has been far from perfect. Young newcomers to Sweden are often placed in special care centers along with other people who have fled from their homeland. This leads to a situation where newcomers to Sweden do not have an incentive to learn the Swedish language as the people around them do not speak it either.

1.2 Problem solution

With this information in mind, we decided to come up with a solution that is both accessible for newcomers to quickly adopt into their life and that also adds an incentive to learn Swedish. Another key factor is the ability to create interaction between people already familiar with the Swedish language and culture and those who aren't. The solution is a mobile application for newcomers to play games that involves them learning Swedish and in the process earning points and recognition for doing so, as well as the ability for people more familiar to the Swedish language to contribute with games of their own.

1.3 Method

To organize the development and have a product to show-of to our supervisors, who want to see new features in the application each week, we used an agile development framework called Scrum. In this project Scrum you have the Scrum Team, Business Owner and Stakeholder. We are the developers and do not really have any Business Owner and as Stakeholder we have the instructor/supervisors and county administrative board, though they did not have any specific restraints on us more than the fact that the application should be used to learn Swedish. More on this in chapter 5.

2 Application of Scrum

Even though the Scrum methodology comes with a great set of tools, the team are always the ones deciding which of them to follow and which not to. Some of the mindsets and tools of Scrum are non-negotiable, eg. the iterative process and constant feedback from the product owner, but others such as pair-programming and meetings policies are not. The following parts will describe how we choose to work and why we did so.

2.1 Roles, teamwork and social contract

In order to avoid any conflicts within the team we sat down early on to create a social contract which we were all to agree upon and follow. It mainly involved three domains. To start off, the contract stated policies as for how often meetings should be held, how we should write our agendas, as well as the rules of absence. Secondly, the contract involves guidelines when it comes to the working process. This includes the responsibility to ask a fellow team member for help if you're stuck and our focus on the quality of time rather than quantity. Finally, the contract also involved general relations and communications topics such as "no silly questions".

As for roles and teamwork, the only thing constant was our Scrum master Adam. He was the natural team leader from the start, and based on the premises that he was one of three team members who live close to the campus, this was an unanimous decision. Other than our scrum master, no other roles or specific responsibilities were set out. This meant that no single person was responsible for a function or a class, rather every single team member was involved in the entire project. The reason for this are quite simple. By not only knowing a small part of the application but instead cooperating on every single piece, we promoted the engagement into, and knowledge of, the application for every member of the group. This kind of team structure remains efficient in such a small project. Any bigger project would most likely require more specific work areas for each team member.

2.2 Used practices

We at some times used a method called "pair-programing" where two or more people programmed on the same computer (this also means that one person got all of the code rows in git). The advantage of this over traditional programming is that the two programmers both can have their input on the work at the same time raising the initial code quality, possibly eliminating the need to rework functions that are already complete.

We also followed general object oriented- and procedural practices.

We ended each sprint with a meeting where we discussed the last sprint, how well it went, how the workload was, created new user stories and assigned new user stories. We let each member choose their own stories to implement with respect to how much time, will and competence each member felt they had the coming sprint.

2.3 Effort, velocity and task breakdown

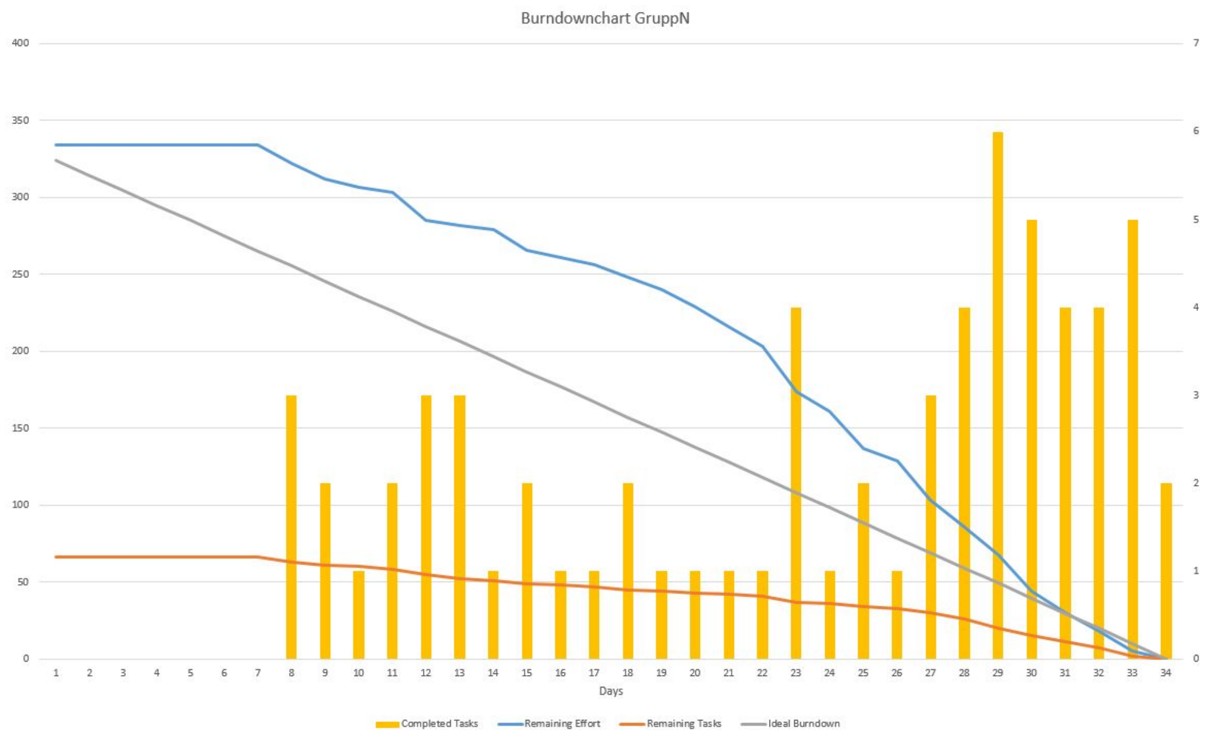
We preferred using short sprints in order to maximize output. At the start of the project, we had a sprint length of one week. However, we quickly realized we had to shorten it to raise the efficiency. The sprint length was shortened to 3-4 days for the rest of the project.

The reason we ended up choosing shorter sprints was because if the user stories assigned to a developer were too short and were finished early, we did not want the developer sitting around waiting for the sprint to end before being able to start on something new. Therefore, the short sprint length increased the efficiency of the team. The shorter sprints also made our group more connected since we had more sprint meetings. With more meetings came more opportunities to ask any questions one might have, and ensured each member was continuously kept up to date on the code.

Velocity and effort was discussed in the early sprint meetings, and was adjusted accordingly.

Each team member got to choose the effort for the user story as they assigned it to themselves, as we all had different experience in working with Android and Java. It would not be fair for the experienced developers to decide the effort of each user story, as it might take much longer for a novice developer to complete.

2.4 Burndown chart



3 Sprints

The sprints were good at making sure everyone did their part before the next sprint as you then can not procrastinate much since you would not be able to finish in time and the next sprint almost always built on what was completed the last sprint.

As can be seen in our burndown chart, we all wrote a lot more code in the last few sprints. This was because of the fact that we wanted all the necessary features to be done for the prototype unveil.

At the start of the project, we did not really have an idea of what the completed application would be. As the deadline was getting closer, we were able to cut off any features that we knew we wouldn’t have time to implement. The net output of the last sprints reflect the finishing and polishing of features that we decided were necessary for the unveil.

There was a turn point as we saw the deadline approaching, where we were able to pin down *all* the remaining work into user stories. Having everything up in the backlog from that point until deadline really put the team into gear.

Anecdotally, the team was largely unmotivated for the first half of the project because of how unclear the Scrum workflow made it feel. We didn’t have an idea of how much we would be able to add to the application in the short time we had. The entire first half of the project, it felt like wandering through a thick two-sprint fog, not knowing where the end would be, or what exactly we were working towards. Again, however, as we got closer to the deadline, the “Scrum fog” cleared and we were able to restructure and regained motivation to work harder towards the end.

3.1 Reflection on sprint retrospectives

As mentioned, the first sprint was very unproductive and led to us almost wasting a week worth of time. This was both due to our initial sprint length of one week and our mentality still largely being that of a waterfall workflow. We focused a lot on sketching a “complete” user interface before even having a running application. This led to the entire team struggle with a task that could not be finished since we didn’t have a running application to test it on.

It was at this stage that the first sprint retrospective gave us meaningful insights into what went wrong. We tried to create a solution in one step instead of incrementally learning new things and exploring what could be better. A conclusion was made that we needed to break problems and tasks into finer details so that we could incrementally improve it rather than trying (and eventually fail) to do it right the first time. Unfortunately we missed a great opportunity by not documenting this part of our meetings as it would’ve now been a great resource to see how not only the application developed but our process.

3.2 Reflection on sprint reviews

Being constantly involved in the coding and development of an application causes a certain biased opinion on what is functional and what is in need of improvement. We therefore decided not to review the current condition of the application in terms of usability and features missing during our sprint meetings, but instead weekly when we met with our supervisor.

Doing sprint reviews together with the supervisor gave us very valuable input and opinions from someone not directly involved in the development. Even though this limited our sprint review to only one occasion per week, we believe we chose quality over quantity in this case.

4 Using new tools and technologies

Going into this project, everyone agreed that we had a good understanding of the tools we were going to use during the project. Hence, we didn't discuss how we were going to use the tools before the project started. However, this quickly proved to be a mistake, since we all had used the tools in different ways in earlier projects. During the early parts of the project this caused some mishaps.

One example of this is with Git, which some of us were used to having different branches including a Development branch and a master branch (also known as "Git Flow"), whereas others prefered only using the master branch and rebasing for such a small project. This created major, sometimes poorly resolved, merge conflicts which hindered our process. But as soon as we discovered the problem, we discussed it during the next Scrum meeting and quickly decided to use one of the methods. From that point on, there were far less problems involving Git as a tool.

5 Reflection on the relationship between *prototype*, *process* and *stakeholder value*

The stakeholders and/or product owner could be viewed as the supervisors, instructor or the county administrative board but they did not have any specific restraints on us more than the fact that the application should be used to learn Swedish. Since the supervisors were the one who had feedback on our design decisions they were probably our closest "stakeholders". Though they gave feedback, they had no vision of how the finished product should be or what it should do specifically. Instead we decided the direction, look and feel of the product ourselves, which meant that we were the stakeholder and product owner and so we made all decisions on the prototype and process.

The transparent development and communication between developers and stakeholder proves the strengths of Scrum. As we had a prototype that clearly evolved and changed after each sprint, the stakeholder was able to see any changes that had been made and comment on them. We believe that the stakeholder value the agile process conveys is its biggest strength. Unfortunately, the stakeholder we had in this project was not the product owner. This rendered stakeholder value largely useless, as the supervisor only gave their inputs and opinions rather than commands or requirements.

6 Earlier practices

6.1 The LEGO practice

As our first introduction to scrum and agile workflow overall we participated in a workshop involving LEGO and building a LEGO city. During this quite chaotic first intro to scrum we learned a few valuable lessons, like the importance of splitting up tasks on different team members, so that everyone has a defined responsibility and knows what to do without having to care about how the other team members are solving their tasks (with the exception of someone asking for help), which is something we got quite good at the further the project went.

Another lesson we got from the workshop was the importance of estimating the time something would take. To say that we were time optimists would almost be a an understatement, and when the time started to run out we rushed out an half assed product which wasn't at all what we or the product owner at all had in mind. This taught us both the importance of correctly estimating how much you can handle during a single sprint and that there is no reason to stress out a feature at the end of a sprint, since that often leads to more work when you later have to redo everything the correct way.

Although we managed to identify the time estimation as a potential problem we didn't handle it very well. We never discussed how we were going to assign realistic time estimation to task and we never did any followup to see if our estimations were, in fact, realistic. If we instead had used our experience from the workshop we might have had an easier time during the actual project.

6.2 Half-time Evaluation

During the halftime evaluation we identified some problems we encountered during our first 3 sprints. Mainly that the short sprints combined with small user stories hindered our ability to create a good and stable codebase, which resulted in a lot of spaghetti code which had to be rewritten in large parts during later, due to the shortsightedness of each sprint. We also felt we had next to no communication with any product owner, and that it was also quite unclear who really was our product owner in the first place.

To combat the lack of good code structure we said we were going to try and have better communication within the team to try and establish some sort of shared vision on how the code should be structured. This proved to be easier said than done, mainly because of the time constraints. In an ideal situation we would have needed to halt the development during

1-2 sprints to only focus on rewriting the code we already had. But due to the need to have somewhat of a finished product by the end of the project this was simply not possible. Given more time this would be the first course of action we would have taken.

The communication with the “product owner” improved after the halftime evaluation, where we had discussions with the students from the ID master program and participated in some workshops where people were able to test the application. However we still don’t have a clear picture of who our real product owner is, thus we didn’t have any way to actually contact anyone who was intended as the end user of our application. This problem is discussed further in section 5.