DAT255

Reflection Report

Group: Zweigbergk

Product: SpeedSwede Android chat application

Team members: Andreas Carlsson, Kevin Chen Trieu, Petros Debesay, Felix Engelbrektsson, Jonathan Krän, Robin Lilius-Lundmark

1. Application of scrum

1.1 Roles among team members

There was at no point in the project any hard-set roles assigned among team members. No one was explicitly given a leadership role, and the role hierarchy overall remained flat throughout the project. This worked well initially and ensured that each team member was equally able to voice their opinions. Further into the project however, issues with agreeing on and failing to adhere to decisions would occasionally surface. Many of these issues could have been either avoided entirely or handled in a better fashion.

One way of going about solving these issues would be to make a team member responsible for enforcing adherence of team decisions. Unfortunately though, doing so would skew the relative power among team members. Thus, a better approach might be to impose a time restriction on decisions, so that the team has to reach a decision within a given amount of time. In cases where the team is unable to reach a mutual decision, the idea supported by the most team members would be chosen. This would ensure that every brought up matter would be decided upon.

Moving on to the issue of decisions not being followed; assuming that the team member not following the decision is operating under the best of intentions, he/she must believe that the agreed upon decision is no longer the best decision. Therefore, require him/her to explain the reasoning behind not following the earlier decision. Perhaps newly acquired insight has changed the best course of action. Moreover, team members should of course ideally gather the group and propose a change of decision before going rogue and dismissing a team decision.

1.2 Teamwork

The team has definitely not been working as a coherent unit throughout the project. One team member spent large amounts of time tinkering with the project outside of the expected 20 hours per week, performing major refactoring in spite of objections from other team members. Said team member found the development to be much too slow for what his expectations on the finished product were and therefore took it upon himself to accelerate the development. As a result of this, the codebase received several overhauls during the course of the project which made it difficult for other team members to stay up to date with the codebase. This in turn made them less productive since the code kept changing, with new unfamiliar abstractions popping up every other day.

The core problem here appears to be a difference in the expectations that each team member had coming into the project. Some team members were content with the present pace of development, while others felt much more inclined to really work towards developing a good product. This discrepancy stems from lacking communication

at the very start of the project. During the first meeting, every team member should have been asked to describe their expectations and goals with the project to make sure that, coming into the project, there was an understanding of how each team member would approach the project. These different approaches could then be combined into a mutual idea of the goals of the project, which could then perhaps somehow be added to the social contract.

1.3 Social Contract

Not much thought was spent on the creation of the social contract which was haphazardly put together. While some of the things that ended up in the social contract were followed throughout the project, the reason for this was not the social contract itself but instead that those things were already being done by team members before the project. The social contract ended up being just another document collecting dust, rarely if ever referenced during development. Looking back at it, several points were not being followed during the later stages of the project.

Hence, to make use of a social contract, there are two things that seem to be necessary. Firstly, the contract must be properly designed so that it actually provides value for the team. Secondly, it has to be accessible on a daily basis at the very least, preferably posted on a wall or similar in the working environment. However, if such walls are not available, like in our case since we were working in different places every day, it should be placed somewhere else visible to team members. One possible, albeit perhaps a tad extreme, solution would be to require each team member to use it as background for their computer.

1.4 Used practices

1.4.1 Pair Programming

The idea of pair programming was brought up by team members on several different occasions during the project, and most of the team felt it would be an interesting technique to try out. The initial plan was to assign two team members to each user story who would then be able to utilize pair programming as they saw fit. However, pair programming only ended up being used on a few separate occasions.

Having said that, team members who had previous experience with the development tools spent a lot of their time helping others solve issues, and this was erroneously referred to as ‘pair programming’ during the course of the project.

There should have been a better understanding among team members of what team programming was since there was initial interest in putting it into practice. It would likely have proved beneficial to the project, since pair programming done right improves the quality of code as well as makes team members more comfortable with different parts of the codebase, the latter of which has been somewhat of a struggle during the project.

1.4.2 Daily Scrum

Daily Scrums were held almost every day. Each team member was asked to specify what they would be working on during the day, and this was then documented. However, this documentation was rarely if ever looked back on, and as such the daily scrums were not utilized fully.

1.5 Time distribution

There were no explicitly established time requirements within the team. A room was made available for working almost every day, and team members were encouraged to be present so that everyone would be within easy reach. Most team members would show up almost every day to work with the rest of the team, while some chose not to. It was rather difficult to gauge the time put in by team members who were usually not working in the same room as the rest of the team. Requiring team members to log their hours of work every day would have helped immensely in estimating the time distribution among team members.

The majority of team members would only work during their time in this room when the whole team was gathered, and then consider the work on pause until the next time the group gathered (usually the next day).

The goal was to divide the user stories in a way such that each team member would need to put in a similar amount of working hours each week. This was made easier thanks to the fact that every user story had an effort associated with it. Yet the time spent by team members was not at all uniform. This could be attributed to differences in knowledge (some team members had not researched certain areas enough to be able to contribute therein), but also to the fact that some team members would more or less only work while being with the whole team. There should have been more emphasis within the team on the importance of familiarizing oneself with the new tools. This would likely have reduced the negative impact that lacking knowledge caused.

There was a general feeling among team members that the effectiveness of the team diminished as the project progressed, likely due to the increasing complexity of the codebase as more features were added without any real attempts at simplification. Heaps upon heaps of refactoring was performed, but it often had the unfortunate result of making the code more complex, rather than simplifying things.

1.6 Effort and velocity and task breakdown

It was apparent that all team members were new to Scrum when we held our own first weekly sprint meeting, and pretty much since. Even though we had a lot of insights with us from the lego exercise, just using this knowledge proved a tremendous challenge.

The first warning came when our first big task breakdown took much longer than expected, which in all honesty may have resulted in an eventually lazier approach as time went; there was so much to break down that we did not give every item as much afterthought as was needed.

To just get everybody to agree to a specific velocity was difficult enough. The number we decided on was decent enough, but we really could have been more specific in exactly what each unit of velocity stood for, since this would have helped us a lot to have a united mind when deciding task efforts.

Some of our task efforts ended up a bit offhandedly set, and added to this was the diffuse velocity, as well as our shallow understanding of how exactly we would put our vision into a concrete product. All of this resulted in tasks that in some cases were greatly underestimated and overambitious.

So our first sprint featured all too big stories, of which more than half of the team was assigned to the same one during most of the sprint. We duly recovered from this in the next sprint by instead overestimating, and ended up with a sprint planning where almost every story had been solved the same day.

Needless to say, we were struggling to get the planning right.

Our planning skills progressed slowly and peaked in the third sprint, where efforts seemed to get a bit closer to reality. When the panic of final demonstration came over us in sprint four, however, almost all planning went to hell, and we just grabbed tasks as best as we could to finish it all. Admittedly, this last resort was partly due to the overambitious scale of the project, but mainly it had to do with our poor planning. For example, if the planning in previous sprints had been more successful, as well as if the team would have put more energy into the last sprint planning, there's a good chance we also would have put more trust in it and ended up actually following the plan. It wouldn't have hurt to have kept our heads cool throughout the last sprint, though.

One thing we realized early is that we, despite our first efforts, could split many big stories into smaller ones, to improve the prospects of our planning. Although we tried to implement this knowledge in the coming weekly sprint meetings, it still really feels like we could have put so much more effort into it; we still had some big stories in the end that was ungainly and hard to break down on-the-go.

On a side note, another thing that affected our planning, was the aforementioned team member that not only did assigned tasks from our backlog, but also worked a lot on the side, on big chunks of functionality that more or less wasn't present in the backlog at all. This resulted in much work going on off the chart, and it being impossible to track either how much or what kind of impact it had on the rest of our planning.

All in all, it really feels like we learned a lot, and mostly about what to avoid when planning scrum. Apparently, it is really important to give the planning the time it needs, and not skid over significant parts, just because we're feeling tired or lazy. It's also important to keep looking at the planning and keeping it up to date (if perhaps somebody is sick or a huge chunk of stories gets done beforehand), so that anarchy is held off bay.

Many, if not all, of our problems in planning could probably have been controlled if we would have thought to look back to earlier documentation and previously discovered issues, to make sure we did not remake our mistakes. As it were, many of our realizations and learnt lessons was forgotten until next retrospective, which, of course, is far from optimal.

2. Reflection on the daily scrum

We had a daily scrum meeting every weekday, either conducted in person or in the project's Facebook group. If someone was missing from the daily scrum, we made sure to note that down in the Facebook group.

Overall, we feel like we used the daily scrum pretty well, everyone had a hum about what other people was doing, which is important. We also set daily tasks for everyone, so it was clear what everybody's mission for the day was.

What we could've done with the documentation from the daily scrums however, was to look at them during the sprint retrospectives, and get even more information about what we could've improved. It was kind of a wasted resource after the day was done, since we did not use it for anything.

For a future projects we could surely have the daily scrum meetings in a very similar fashion, but additionally also actually use the documentation and draw wisdom from it.

3. Reflection on the sprint retrospectives

Our sprint retrospectives could have been so much better. We focused way too much on coding, rather than analyzing what we could've done better and more efficient.

In the first sprint retrospective, after we finished sprint number one, we all agreed that we had pulled in too many tasks from the backlog. What we probably should've done here is to either lower the overall velocity per sprint or raise the effort of each story. We didn't do any of this however, instead we moved on to the next sprint with the exact same velocity. During the following sprints we did actually reach the goal we set up, but this was only thanks to one of team member working off-hours with the product.

We did however have a sprint reflection after every sprint, and we did get some potentially good information from it. Yet we did not act as much on this information as we could have, and just continued doing the same thing as the week before. A lot of it probably had to do with the fact that we met most of our sprint goals anyways.

There is a lot of room for improvement in this department, and the easiest thing to do would have been that everyone followed what we learned from previous sprints.

4. Reflection on the sprint reviews

We had two reviews scheduled on wednesdays with the interaction design team. This proved successful as they were quick to identify design flaws in our application as well as suggest features they thought were important. Besides these two reviews we planned to have an internal sprint review among team members every friday evening. However, this never came to be.

Not having these weekly reviews turned out to be a detriment to our effectiveness because they were supposed to give us insight into what the other team members had worked on during the previous sprint. Consequently, we were late on discovering inconsistencies in both the functionality and how much team members worked, as well as design flaws, which were only brought to our attention in the next sprint.

One of the reasons that the weekly reviews fell through were that team members were not strictly following the sprint working hours, and instead kept working after the actual sprint had ended. This in turn would render the sprint review less effective because more things would always change before the next sprint retrospective.

Having a designated ScrumMaster would likely have helped with making sure that the weekly reviews were carried out. In addition to this, it is important that the team agrees upon during which time frames it is okay to work on the project.

5. Best practices for using new tools and technologies

About half of the members in the group had never used Android Studio prior to this project. However this didn't cause any major problems since we felt it was a fairly intuitive environment to get up to speed with.

Our user stories were managed in Trello and sorted by priority. At every weekly scrum, we selected the top ones and placed them into a separate "in progress" list. We then further divided these stories into small tasks. Whenever a task was done, it was checked off, and when all tasks were done we moved the story into a list of "Testing". When a story was tested we moved the card to the "Completed" section.

A separate spreadsheet document was used as a burndown chart, which we filled in after completing a user story. This gave us an overview of how far we progressed in the sprint.

We used Git with Github as our version control system. In the early stages of the project, one team member presented a workflow for git. The idea was to keep all the local work on a separate branch and never merge directly into the master branch. But most team members did not follow this workflow, yet also did not feel like it was a big issue. We never had any major merge conflicts, probably because we made sure to communicate what we were working on and often pushing changes to github.

A strict rule not to push anything up to Github that weren't working was agreed on. This was done in order to prevent interrupting the workflow of other members. This was

followed for the most part, but evasive bugs would sometimes find their way onto the main branch. These were mostly minor however, and were fixed shortly after they had been brought to the team's attention.

6. Relationship between prototype, process and stakeholder value

We started out with a scope of application that was a way too big to finish in time. We should have set a more realistic vision from the beginning, so we would not have to scale back so much from it during the sprints.

Our basic function of the program was a pretty difficult task in itself, and to that we wanted to add a bunch of other things aswell. Perhaps we should've chosen a completely different path even from the start.

We did deliver a lot of stakeholder value in most of the sprint, and we got a neat product in the end. However a stakeholder might have expected to get all the things we thought we would be able to produce in the beginning, and they would probably be very upset not getting all these extra features that we had in our vision.

We think the discrepancy between our vision and our product was too big to be acceptable, and clearly the biggest issue was that our vision was unrealistic.

7. Relation of your process to literature and guest lectures

7.1 Literature

7.1.1 Scrum XP from the trenches – We read this book prior to starting and thought we had a pretty solid understanding of Scrum and how to use it in real life. Nothing could have been further from reality. We all did a lot of mistakes and fell into pitfalls during the project. When discussing this afterwards we came to an agreement that Scrum is something you have to learn by doing.

7.2 Guest lectures

7.2.1 Volvo

Volvo is a large company with lots of traditions and according to the lecturer it was sometimes difficult change their way of doing things. Historically they had used the waterfall method but tried to move to a more agile way of doing things. The lecturer took up some of the difficulties of introducing new ways of working in such an old organisation.

A key take-away from the lecture was the importance of getting the teams to communicate to reach success. In our project we mostly tried to sit in the same physical space when working. Instead of booking a meeting beforehand we could discuss problems arising in a relaxed way. This is a luxury in comparison with a company that have tens of thousands of employees and lots of subcontractors.

7.2.2 Spotify

In comparison with Volvo, Spotify is a completely different kind of company. It's a fairly young company that embraces the agile workflow and encourages teams to be independent units. This way of working has many perks, a major one being that teams could choose their own way of doing things and put code in production quickly.

Our way of working is similar to the Spotify way of doing things. We move fast and break things. Our team size is small, similar to the size of a squad at Spotify, and we also tried to continuously deliver user value in small increments on a sprint to sprint basis.

8. Evaluation of D1A and D2

8.1 D1A - Lego scrum exercise review

The very first thing that is brought up in the review is that we shouldered much more work than we should have. We discuss how our time assessments for stories were far too optimistic. This is something that we felt improved as the project progressed, albeit not by as much as we perhaps would have liked. As the project was nearing its end, we still had issues with stories spilling over into the first few days of the coming sprint, despite our efforts in reducing the amount of stories that we brought on each sprint.

The main reason for this is likely that we were not following up on the previous retrospectives as well as we should have. We kept looking forward, pondering on how to continue our work, without seriously contrasting it with issues from earlier retrospectives. A key lesson here is that looking at and referencing earlier documentation is of great importance. If you do not recall and consider your past mistakes, there is nothing stopping you from making them over and over again.

Furthermore, we realized that poorly specified tasks were difficult to complete since many assumptions have to be made. This is something we feel has improved over the course of the project. Our stories at the end of the project were more detailed and specific than those early on.

However, we have still been making assumptions, especially about our intended users. We write in the review that we will try to maintain an ongoing dialog with the product owner, but we really have not. It would have been of tremendous help to actually talk to

our intended users and ask them what they wanted instead of playing the guessing game, hoping that we hit jackpot.

We also bring up that dividing the group into sub-groups seemed to be an efficient way of working. This is something that we have implemented, where we assigned 1 to 3 team members to each story. And while it did reduce collisions between team members in terms of who works on what, it was still common that the work of team members clashed. Although as the project progressed, we felt that we had less of these clashes, likely as a result of us improving at breaking down stories and their related tasks.

8.2 D2 - Halftime evaluation

We had a problem with the user stories being way too big. After D2 we started to break down the tasks to much smaller tasks. This made things a lot easier.

Testdriven development was something that we wanted to implement to a greater extent after D2, because we thought it might increase our productivity, force us to write better code and not having to manually test everything. We did not start using it more, though, which might have been a mistake. As an example, a lot of time has been wasted compiling the program, which perhaps could have been avoided by writing unit tests before, and developing the functions from that point.

The halftime review and meeting up with the interaction designers again made us think a lot more of the actual interaction design. We had to take into consideration that we might accidentally offend some cultures with minor things, which we hadn't thought of at all. We also tried to put a lot of focus in making the application as simple to use as possible.

We ourselves thought we managed to do a pretty good job with usability. When we showed our application to the public though, we found out it was still far from intuitive to use. We think a huge lesson was learned from that. It is clear that we should have shown our product to outsiders much more often to get input on how others experienced our application.

9. Burn-down chart

The burn-chart only takes into consideration the stories we actually planned to finish during our four sprints, including those we did not finish.

On the other hand, the chart does not include all the stories we created at the beginning of the project. For instance, the quiz game between two chat participants that we originally wanted never made it into the application. The chart also does not include the big amount of work done outside of project hours, that never were formulated into stories, mainly because the lack of documentation of this makes it impossible to estimate stories and their efforts afterwards.

We thought this way of doing it visually showed our progress most clearly.