Course: Software Engineering, DAT255
Examinator: Jan-Philipp Steghöfer
Authors: Adrian Lindberg, Axel Ljungdahl, Erik Jansson, Jonatan Nylund, Jonathan Sundkvist, Sebastian Nilsson och Matthias Andersson

Reflection report

1 Application of Scrum

1.1 Roles, teamwork and social contract (relates to D1B)

The very first meetings as a project group consisted of the group members asking themselves and each other what we wanted to achieve during this course, both individually and as a group, the objective being to find a common ground and a level of ambition that we all could agree upon in order to avoid eventual conflicts. With the individual ambitions in mind, the idea was also to hand out specific roles to the group members in order to address their desires to work within a specific area (e.g. graphical design or database structure). All this was also explicitly written in the form of a social contract that would state the agreed-upon rules, guidelines and roles, as well as what to do in the case of a member of the group not following them.[1]

Fortunately, the group instantly realised that everyone had the same vision of how much effort was to be put into the project and what would be demanded of the members in order to realise the group's vision. The group also had the advantage of having a good mixture of experience and expertise between the members, allowing the more adept ones to take on more supervising roles in order to create opportunities for the less experienced members to work within an area of their choice. However, it is pertinent to point out that even though roles were handed out, these were not in any way set in stone but rather treated as guidelines when distributing tasks at the beginning of every sprint. This becomes apparent when looking back on the social contract, since most members ended up only partially working with their requested types of tasks. In hindsight, this also seems like a natural development of events in line with the techniques used when working with the Scrum development model, such as splitting up user stories into thin, vertical slices. This essentially meant that every member had to not only, for example, work with the graphical aspects of the project, but to take all parts of the application into consideration. The lesson we learned was that, when working with Scrum, one has to design the vertical slices truly to the Scrum model and its principles and not let individual members' wishes affect this process. For further discussion and reflection on the topic, see sections 1.3 and 1.4.

1.2 Used practices

Deciding on how to approach the task did not prove to be more of a challenge than agreeing on a common vision on what was to be done. The initial plan was to code in pairs, not only because the group didn't believe that the project was big enough to occupy seven people working on separate features but also as an attempt to satisfy the members' various requests of roles.

[1] https://github.com/drualsk/RefugeeMap/blob/documents/Socialt%20kontrakt.md

The idea was also to follow a rather strict schedule with an early monday gathering to begin the week's sprint and hand out assignments, daily stand-up meetings on Tuesdays through Thursdays as well as a longer meeting on Fridays to properly merge the past week's features. The daily meetings were to be used as a way for all members to stay up-to-date on what the others were currently working on, and were intended to be as concise as possible rather than a platform to discuss complex problems in detail - such things were to be discussed separately between those involved. Code review was also something to be done by the end of each sprint in order to maintain a high code standard.

The general structure of the sprints and their meetings was a successful aspect of the project and something that has been crucial to the workflow.
The Monday meetings served as a distinct end to the previous sprint (in case some additional fixes were made during the weekend) as well as a natural starting point for the next one. During these meetings, we performed a sprint retrospective followed by the distribution of user stories for the upcoming week.
The daily meetings took place three times a week as planned, however, the group had to take into account the different schedules among the members as some had multiple courses or part-time jobs. This was easily solved by the possibility to attend the meetings via Skype, or, if a member was unable to do so, to simply send a message to the group with relevant information concerning his progress.
On the other hand, the Friday meetings were not executed fully as planned - they did not become the distinct endpoint of every sprint due to aforementioned schedule-related conflicts which occasionally resulted in members having to complete their assignments during the weekend. While this did not prove to be a significant drawback to the workflow, it did slightly counteract it - in the end, it instead made more of an impact on the prototype itself, as there had been no proper sprint reviews. The group and the project would have gained significantly on scheduling regular sprint reviews in order to produce maximum customer value in relation to work effort. For more on this, see sections four and six.

The intent to code in pairs was quickly put aside due to differing schedules and in order to work more efficiently, since the group realized early on that the project would require everyone to work on separate features. Code review was put aside almost completely. Since the group used a workflow based on feature branches and pull requests, the Git administrator was required to perform basic code review before merging in a feature. This is something which would have been handled differently if we were given the chance to redo the project, since merge conflicts became more frequent as more code was added. In hindsight, there were some features that should have been lowered in priority in favor of thorough code reviews.

1.3 Time distribution

The group strived for a twenty-hour work week per member. The time spent on each sprint was quite evenly distributed among group members, the possible exception being the Git administrator who briefly evaluated the code in each pull request, resulting in a slightly higher workload.

To use the allotted sprint time as efficiently as possible, effort was made to ensure that each user story was as modular and separate from the rest of the application as possible. The idea was to remove the problem of needing to wait for a fellow group member's code in order to be able to continue one's own work, and this was largely accomplished in tandem with the “thin vertical slices” which were strived for during the planning of each sprint - the proper division of features resulted in decreasing amounts of dependency issues.

Despite the aforementioned assignment of roles, areas of responsibility ended up being frequently switched, which forced the group members to wander outside their comfort zones and made the project a more valuable learning experience. This also made each group member more dispensable and easily replaceable in the event of sickness or other unforeseen circumstances. This lax attitude to the assigned roles was mainly applicable to the Android programming as the database code and user experience design were handled by the same group members during the entire course of the project.

Arguments can be made both for and against the strict assignment of roles. On one hand, having set areas of responsibility means minimizing the time needed to gain insight to the written code and APIs. It also allows for group members to really specialize in a certain area, producing well-structured and generally high-quality code. On the other hand, this specialization makes the group less flexible as a whole and can completely throw off the planning of the project in face of sudden changes in personnel. When dealing with an intensive and brief development phase as in this project, flexibility can be an extremely useful asset. Specialization can also make a vertical development process more difficult, as a single developer may lack the knowledge to execute a full vertical slice.

Evidence for certain points can be found in the source code for the application, the general rule of thumb being that the fewer people having worked on any given part of the code, the cleaner and more well-structured it became.

1.4 Effort, velocity and task breakdown

In the project’s earliest stages, a development velocity of 140 function points was established - since each of the seven group members was expected to work twenty hours per week, one point was equivalent to roughly one hour of work by one person. However, the group failed to account for the roughly two hours of lectures and three or four hours of meetings every week, which resulted in the group spending more time with this course than intended. This was realized more than halfway through project, and instead of adjusting the velocity or redefining the convenient meaning of one function point, each task was overestimated in order to compensate for the “incorrect” velocity, leading to less rigorous and less precise task breakdowns.

Task breakdowns were a struggle from very beginning of the project, the main challenge being consistently adding user value to the application as well as evaluating what the definition of “user value” actually was. Was a bug fix considered valuable? Did improving code quality and readability imply value to the customer (and if not, how was it related to the backlog)?

Another difficulty concerning task breakdowns were their verticality. Certain improvements to the user experience of the application did, for instance, add value to the application but were not considered vertical due to the fact that they did not touch the backend code of the project. At multiple points in the project, the design supervisors were shown features that didn't have a connection between the frontend and backend (despite both being completely developed) or certain features being present in the backend but not yet having surfaced to the frontend and thus presenting no customer value. Another pertinent issue was the inconsistent size of tasks - one tell-tale sign of a backlog with this problem was a large variance of function points between the user stories. One way to tackle these issues was to have a certain divide between user stories and other miscellaneous improvements. The user stories' additions to the customer value of the application should be doubtless, whereas miscellaneous improvements (such as code reviews or user experience improvements) may not bring immediate value to the customer or be strictly vertical but may be beneficial to the development in the long term. The prioritization of these improvements should be discussed in order to thoroughly determine both positive and negative effects. The group often neglected the improvements "invisible" to the customer in order to maximize the customer value added to the minimum viable product during the four-week development.

It was noticed that the effort needed for a given user story tended to correlate with the product value added by the user story. Sometimes, asking "how much will this user story improve the application?" instead of "how long will this take?" offered new insight during the evaluation process. It was also discovered that smaller tasks were almost always easier to work with and as such, effort should have been made to fully dissect each user story.

Another valuable guideline for dealing with user stories was to set up the backlog according to the **INVEST** criteria. Discussing user stories and their relation to *INVEST* ensured that every backlog item was a small, modular cog in the larger machinery of the application and allowed for a more structured project. This tactic was adopted to create order in the unstructured backlog present in the early stages of the project where the group compared backlog construction to "fumbling in the dark".

2 Reflection on the sprint retrospectives

The plan was to have a sprint retrospective meeting after finishing a sprint. This was held on Monday mornings before the start of the next sprint, the goal being to identify problems with the work process, find solutions and not let eventual disputes or confusion remain in the next sprint.

In turn the team members were all asked three questions:[2]

- "What do you think went well the last sprint?"
- "What do you think went wrong with the last sprint?"
- "What do you think we need to change this upcoming sprint?"

This way everyone had the ability to voice their opinions, and discussions could revolve around what to do differently to improve the workflow.

The sprint retrospectives were very useful for the team and fulfilled their purpose. Some of the workflow issues that didn't make it into the daily meetings could be addressed during the retrospective.

The team did, however, miss out on the sprint review meetings, which would have been suited to have right after the sprint was finished (for further discussion, see section 6.0). As a result, some of the issues that could have been addressed during the sprint review (such as technical questions, questions regarding current state of the application, etc.) leaked into the sprint retrospectives, which caused the team to have less time for the subsequent sprint planning and thus also less time to think about assigning function points to user stories.

One simple solution would be to have a sprint review on another occasion, separate from the sprint retrospective, such as on Fridays. If a distinct difference would have been made between discussions regarding the current state of the application and workflow itself, the retrospective discussion would have been less cluttered.

3 Documentation of sprint retrospectives

Documentation from the team's sprint retrospective meetings can be found in the project's Git repository on GitHub, under the **documents** branch.[3]

[2] https://www.scrumalliance.org/community/articles/2014/april/key-elements-of-sprint-retrospective
[3] https://github.com/drualsk/RefugeeMap/tree/documents

4 Reflection on the sprint reviews

Although the group didn't explicitly schedule time for sprint reviews, it occurred "in disguise" once a week during meetings with the design students. On these occasions, they gave input and feedback on the product presented to them and, in some ways, acted as product owners. However, this was hardly an optimal way of performing the sprint reviews - as the meetings took place in the middle of the week, the group could merely present semi-complete versions of the application. Unfortunately, the lack of sprint reviews was not noticed until late in the process and therefore it was decided that it would be seen as a lesson learned to put more emphasis on in the future, in which case the review would explicitly be scheduled as a separate event and be treated as an essential part of the work process (advantageously placed at the very end of each sprint).

5 Best practices for using new tools and technologies

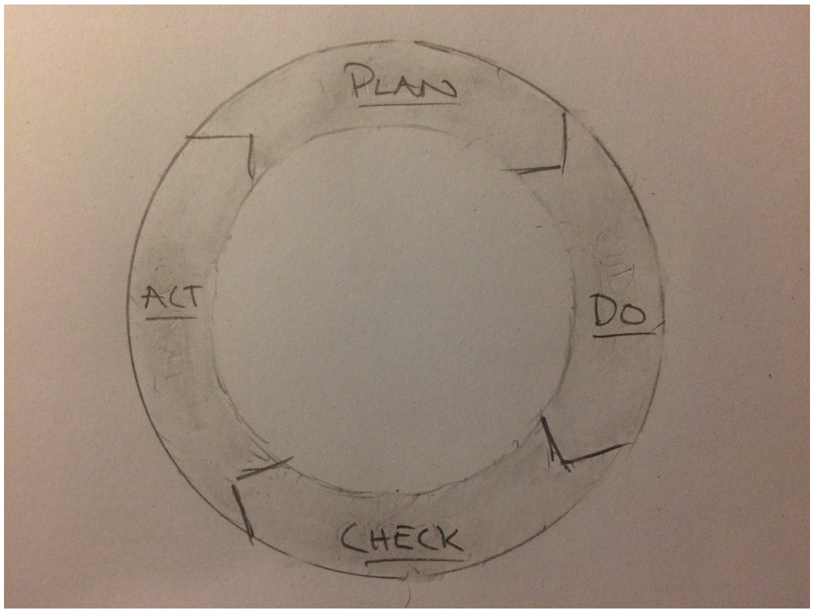
The guidelines set up early in the project mainly concerned the Git workflow and the Android development suite. The Git workflow utilized feature branches, each generally corresponding to a single user story. Upon testing and subsequent completion, the feature branches were submitted as pull requests to the Git administrator. The pull requests were briefly reviewed and, unless an obvious problem was found, the code was merged with the active sprint branch. This method seemed cumbersome and needlessly complex to those unfamiliar with this way of working with Git, but all group members were eventually convinced that this way of working led to a more structured project.

One thing that was lacking was manual testing of the user interface. Whereas the quality of the model code could be automatically tested using *JUnit*, the GUI-specific portions of the code had to be manually tested - this was an unstructured way of working and led to many bugs sneaking past tests and finding their way out of the feature branches onto the semi-stable sprint code. A more ideal way of working would have been to have a structured way of manually testing the GUI code; a sort of checklist to run through on every feature being added. This could have been further streamlined through the use of automated GUI testing through the use of scripts.

There was little prior experience with Android development in the group, which greatly affected the function point evaluation of the backlog. The unfamiliarity with the technology was accounted for in that the effort of user stories involving learning a new part of the API were generously overestimated. This turned out to be a sound tactic and as the project wore on, the estimations became more and more accurate even in the face of unfamiliar problems.

6 Relationship between prototype, process and stakeholder value

In order to have a good and efficient workflow one needs to find a balance in the following aspects of a process:



The concepts of prototype, process and stakeholder value are intimately tied together. By having a well-structured workflow with continuous user testing and input from the product owner, it is easier to realize what works and what does not. The prototype can thus be considered the result of how well the process worked and how well it incorporated the stakeholder's value.

The initial ambition was that all aspects of the process should be taken into consideration during the whole process, since they all play an important part in the successfulness of the final prototype. Three of them were continuously a part of the process. For instance, every Monday, both the "conclude" (labeled "act" in the image) and "plan" part of the process was realized. The "conclude" part consisted of having a sprint retrospective on last sprint's strengths and weaknesses, and the "plan" part was based on this analysis. Furthermore, the "do" phase was less of a problem as time had been spent on setting a common vision of the project which everybody could agree upon as well as having a social contract for the group.

However, despite the initial ambition, one can conclude that one of the aspects was clearly lagging behind, namely the "review" part (labeled "check" in the image), which became increasingly apparent as the project progressed. More reviewing by the group itself would have been desirable, as well as better utilization of those resources available. For instance, the design students were somewhat seen as the product owners and they gave important, constructive feedback on the prototype. However, the benefit of having them testing the application could have been used to a larger extent, as well as actually trying to stay in contact with someone in the actual user demographic (i.e. newly arrived refugees). The group also had the ambition to reach out to organisations and companies at one stage in the process, in order to get their input on the product (both regarding the idea itself as well as the actual prototype), but this idea was never realized.

With this taken into account, one could argue that the “review” part of the process was somewhat weak in comparison to the other parts. One attempt at remedying this, as was advised by the design students, was to create personas, the idea being that these personas would offer insights that the group itself otherwise would have missed when testing the prototype. These personas were created, but not heavily used. They could further have been extended to incorporate companies and organisations (the product’s secondary user group) as well, since no user testing was performed on that part of the demographic.

The conclusion is quite obvious. The overall process, as well as the product itself, lives up to the vision in large, but would have benefited from having more extensive user testing (in any of the forms mentioned). One evident example of how this could have helped is regarding sorting events by at what time they occur - this is something that can be regarded as quite obviously important to the user, but was missing in the application up until late in the process. This was not noted (due to the lack of reviewing) and is something that a typical product owner would likely have insisted on early in the process. More examples of the same nature can be found, and this could be considered a lesson well learned.

7 Relation of your process to literature and guest lectures

The group aimed on having consistent style and following the style guides from the Android API documentation.[4] One example of its relevance to the project was how the communication between an **Activity** and its **Fragment**s should be handled. Other important aspects of the guidelines were how one should set up a navigation drawer and what proportions it and its icons should have as well. This was discussed extensively but not entirely implemented in accordance with the guidelines. The reason for not going completely in accordance with the design guidelines was based partially on a simple judgment call, but also due to lack of experience with the Android framework. Some members in the group had prior experience working with Android Studio, but despite that fact, working with an **Activity** and **Fragment**s ended up being more challenging than expected. Perhaps the Scrum methodology wasn’t ideal in this aspect, since the methodology aims for short sessions that are supposed to deliver customer value. What the group might have needed was to sit down and make sure of how to build the basic frame of the project before starting implementing specific features (which was the case when working with the Scrum method). With this being said, it might not necessarily be a problem related to the Scrum methodology, but rather the group’s interpretation of it.

The course literature was also helpful when writing the project vision. With the literature as aid, it was easy to understand what was expected from a product vision and how it could be of assistance for the group itself during the work progress. By having a well-formatted vision, it became easier to work toward a common goal. Not only did the vision help maintaining focus during the development, but the process of writing the vision itself also had an impact on the project. While trying to narrow down the project's scope as concisely as possible, the group realised that it could not be done in a way that would satisfy the

[4] https://developer.android.com/

definition of a project vision, due to the fact that the project had two different target groups at first, with separate functionality depending on which one was using it, causing the scope to be too wide. In response to this problem, the group decided to focus on one of the target groups only - the newly arrived, rather than the organizations. Narrowing the scope and having it put down on paper has, in hindsight, been critical to being able to deliver a functioning prototype and having a successful project.

The guest lectures were a source of inspiration. A point of particular interest was how Scrum works in real life. One example is how Spotify works with “Alliances”, “Tribes”, “Squads” and “Guilds” instead of the “regular” teams. Similar to Spotify, the group strived for a more strict distribution of roles initially, but in contrast, in a small scale project like this, all member had to work on many different parts of the code. This was done in order to deliver more finely separated user stories and more easily measurable customer value. This adjustment was adopted quite early on in the process, and seemed like a more natural way to divide the work in a project like this. For this reason, it was not changed during the course of the project.

Spotify and several other companies try to show off their environment as being relaxed, where one is free to work with what one wants as long as one delivers before the deadlines. The principle of working freely (to “just get the job done”) was adopted by the group when delegating user stories. A member was generally allowed to work at his preferred pace as long as his work was finished in time for the deadline. During the last sprints, some of the members chose to finish their work after the sprint deadline. This wasn’t due to the size of the workload but rather to the aforementioned relaxed approach. This approach has its drawbacks - for instance, if a member gets sick or is in any way unable to finish on time, the group still needs to be able to do the weekly review of the prototype, which ended up being an occasional problem. Thus, the conclusion drawn was the importance of deadlines as well as the need to strive for less dependency on each individual. If an obstacle would occur that would prevent the individual from actually performing, the group as a whole would stand better prepared if it had a low dependency on that individual (i.e. if the user stories were sliced finely enough and lived up to the INVEST criteria).

8 Evaluation of D1A and D2

Three valuable lessons learned from the Lego exercise was how over delivering features was a waste of both time and money, the importance of communication (not only between the group and the product owner, but within the group itself) and the difficulties of a reasonable time estimation of every feature. These insights were considered throughout the project, but ended up being executed with mixed results.

The group did not end up overdeliver on the desired features, but this had more to do with the lack of distinct definitions of “done” for the features than the members putting down just the right amount of work for them to be considered finished. The group would have benefited from having clearer definitions of when a feature was finished, both to avoid putting down as much excess time on a sprint as the case was, as well as in order to more precisely estimate the time consumption of the upcoming sprint’s features. This problem was closely related to the lack of an obvious product owner, which was also brought up during the halftime review. Who was supposed to determine when a feature was considered finished? If not told otherwise, a software developer will have no problem coming up with additional features to add to the existing ones. While weekly meetings were organized with our designated design student, who delivered useful feedback on the progress, in order to fully utilize their expertise it could have been explicitly stated that the group wanted them to act as product owners and deliver their feedback accordingly.

Due to aforementioned reasons the communication-related parts of the project have been somewhat lacking in relation to what was learned during the Lego exercise. The group had learned that communicating with the product owner was an important aspect of a successful design but, in the absence of a proper one, this part was not properly utilized. Aside from the difficulties when estimating the time for each feature, another trouble caused by this was defining what a user story actually is. Does it always have to bring customer value? Is a bug fix to be interpreted as a user story? These were questions that by the end of the project still had not been straightened out. On the upside, the communication within the group has been far better, thanks to both the daily meetings and the use of an online communication tool which allowed members to express thoughts and ask questions throughout the process. This has been important for the group in order to compensate for miscalculations or misjudgments related to other aspects of the project, as other members were always available for support when needed.