Final Team Reflection

Week 42 & 43

Team members: *Nur Hussein Abdulkader, Tom Bjurenlind, Daniel Cebe, Rickard Gyllensten, Roman Melnik & Christer Sonesson*

Customer Value and Scope

- the chosen scope of the application under development including priority of features and for whom you are creating value

(A) Currently, the application is supposed to give points for the number of passengers in a car. The points can be accumulated in order to earn rewards such as coffee or snacks at gas stations. The idea is to encourage carpooling in order to reduce air pollution and traffic congestion. The priority of features: identify the number of people in the car, award points for the current ride, store/load points and show total, create a store where rewards can be redeemed. Value created for: end consumers (drivers), Volvo Car Corporation.

(B) Eventually, a future version of the app should have following features: a leader board that shows the high score for all users or a group of users, so that you can compete with everyone or in a certain group. Value created for: end consumers (drivers), Volvo Car Corporation, the society (less pollution, less congestion).

(A→B) To get there, we must implement the leader board feature, which would make the app more interesting for people who like to compete with others.

- the success criteria for the team in terms of what you want to achieve with your application

(A) A application with a simple interface that awards points for the number of passengers in the car. The app has a store where the points can be redeemed for rewards by receiving a coupon that can be used at a gas station.

(B) A user-friendly and intuitive application with an advanced user experience and user interface that gets many downloads and many active users who carpool often, thus earning a lot of points and redeeming a lot of rewards. A large choice of different rewards. A cleaner environment and less congested roads.

(A→B) To get there, we must implement new features and an advanced interface, maybe asking users to rate our app in the store. An app with a high rating would gradually get more downloads and more active users. We must also come up with new attractive rewards to offer.

- your user stories in terms of using a standard pattern, acceptance criteria, task breakdown and effort estimation

(A)

    - As an eco-friendly driver, I want to be rewarded for having passengers in the car and possibly for eco-friendly driving, eg. driving with low RPM (high gears) when possible.
    - As a driver I want to see my awarded points on the board that shows the number of points awarded for the current trip and in total.
    - As a driver I want to be able to visit the store page quickly, since I want to spend my points for rewards, and view my inventory (what I have spent my points on).
    - As a team we want to learn about Android API since we need this information to to complete this project.
    - As a team we need a version control system git repository so we can communicate and work with the tasks.
    - As a developer I want to use the simulator signals in order to determine the number of people in the car.
    - As a driver I want to see the items I have bought in an inventory since it is easier to visualise the items.

Our general strategy for handling the above user stories is to attempt to estimate how much effort they require in terms of one sprint (one week's task work) and break them down into smaller features, and these into even smaller tasks, which may then be individually completed during one or two week's work (approximately). Some of our effort estimates proved incorrect. For instance, the coupon store took longer time to complete than we planned. Our acceptance criteria is simply that the user story is met by our implementation.

(B) In a future version of the app, we will have all of the above, plus following:

    - As a driver who likes to compete with my friends or colleagues at work, I want to be able to create a group of users to compete with.
    - As a driver who likes to compete with everyone, I want to see the high score on a leaderboard.

We should be able to estimate effort more easily and have a clearer acceptance criteria for each user story.

(A→B) To be able to estimate more easily, the team needs more experience of app development. When you discover that your effort estimates are incorrect, you gain experience that will help you make a better and more correct estimation next time. To achieve clearer acceptance criteria for our user stories, we should use a checklist called Definition of Done.

- your acceptance tests, such as how they were performed and with whom

(A) Acceptance tests for:

- Working on the coupon store: testing that the core functionality works as it is supposed to, namely that awarded points can be traded for any reward from the store by receiving a QR code that can be redeemed at a gas station.
- Getting simulator signals from the seat belt sensors: testing that the signals are read as they should, that the number of people in the car is calculated,

used and displayed appropriately and that we receive signals indicating when the simulation is over (i.e. engine off).
- Scoreboard for the current trip: visually verifying that the scoreboard shows the correct number of points for the current trip.
- Scoreboard for the total points earned: visually verifying that the scoreboard shows the correct number of points for the current trip and that saving and loading the total points works.

The tests were performed mainly by the team member responsible for the respective task, and also by the entire team when everyone merged the changes. However, the team tests were not done as regularly as they should have been.

(B) In a future project, the acceptance tests should follow a checklist called Definition of Done, and the acceptance tests should be done by the whole team on a regular basis.

(A→B) To get there, the team should create a checklist called Definition of Done for every user story, and the acceptance test should be a compulsory item on the agenda of every sprint review.

- the three KPIs you use for monitoring your progress and how you use them

(A) We have now developed most of the core features and our main KPIs are *team velocity* (task work done each week/sprint), *number of completer user stories*, *quality* (e.g. a bug-free, smooth-running app) and *usefulness* (i.e. is the app solving a problem or providing a benefit). We tried to keep a reasonable pace and complete the tasks assigned for the current sprint.

(B) In a future project, to make the team more efficient, we might add more KPIs, such as *sprint burndown* (the amount of work done throughout the current sprint), *cycle time* (time from 'in progress' to 'done') and *cumulative flow diagram* (a diagram that shows the progression of 'to do', 'in progress' and 'done' over time).

(A→B) To get there, we should study and choose among the many KPIs available in agile development. We could try one or two new KPIs and see if they help us increase our team efficiency, and perhaps add more KPIs later on.

Social Contract and Effort

- your social contract, i.e., the rules that define how you work together as a team (this means, of course, you should create one in the first week)

(A) Our social contract has not changed since it was formulated in the beginning. We still strive to help each other out when there is a need as well as divide the work equally between all team members. We adhered to this contract as far as possible.

(B) In a future project, we might improve our social contract from start by establishing what is important for the team, what people would like to get out of this project, and what we can count on from one another.

(A→B) To get there, in the beginning we need to have a session dedicated to creating this social contract. We need one team member or a different person to act as a facilitator who asks the team and individual members many questions, both general and specific. E.g. “What matters most to you in this project?” or “Is it okay to submit buggy or ‘ugly’ code?”

- the time you have spent on the course (so keep track of your hours so you can describe the current situation)

(A) Like we have during the entire project, we define the hours spent on the course as the hours spent strictly on the project, averaged over all team members. With this definition we have probably spent around 12 hours this week and last week (combined Sprint), so in total about 8 + 13 + 14 + 12 + 6 + 12 = 65 hours (as previous weeks’ work was estimated to about 53 hours). Among this and last week’s hours, most of it was spent as individual work. (The time for writing these reflections is as usual excluded).

(B) In future projects we will probably want to spend more hours on the projects at hand; since our hours stated above aren’t too impressive according to ourselves. The reason we didn’t spend more hours is because we wanted to focus on the core concepts of the course, such as applying Scrum and getting acquainted to working in a “bigger” team in general (and all the tools that come with it to make it work), but we still finished the core tasks we stated in the beginning of the course (so strictly speaking more hours weren’t demanded of us). We’d probably want to spend more hours on implementing further ideas into the application in a future project though (e.g. online leaderboards in this case).

(A→B) In order for us to do this, we will have to set out bigger goals for ourselves, and not decide to settle on the initial core features stated in the beginning of the project, but instead set more hours aside to explore more advanced ideas and features.

Design decisions and product structure

- how your design decisions (e.g., choice of APIs, architecture patterns, behaviour) support customer value

(A) This and last week (2 week Sprint) we finalized the app. We have only used the standard Android APIs and Android-Car APIs, and we have built on top of the foundation work we executed previous weeks. In this Sprint there was more focus on good design decisions; some of the foundation work from previous weeks was refactored and commented to reflect this. In the beginning of the project we didn’t spend too much time on thinking about what would be the best, most maintainable way to implement and integrate each feature. In the final weeks we did however look over our code base and did refactor the most obvious “flaws” in our design decisions. We also added comments for many of our core methods, in order to make it more comprehensible for everyone.

(B) In the future it would definitely be beneficial to think about good design decisions from the very start - since this makes it significantly more maintainable from a developer’s point of view. There’s really no reason not to do it to be frank; I guess we were just lazy, or didn’t think about it at all. Since we only used the standard Android- and Android CAR APIs in this project, it would be interesting to incorporate more interesting and useful APIs in the future; such as Google Maps, MongoDB, etc.

(A→B) To do this we'll have to participate in more projects (hopefully bigger projects, making use of more APIs) and also bear in mind to use good design decisions as well as select useful APIs where we can from the very start. This will significantly simplify working on the project in general, and reduce the time spent on implementing new features. All of this will obviously also highly benefit the customer as well as the product owner.

- what you document and why, by using e.g. use cases, interaction diagrams, class diagrams, domain models or component diagrams, text documents etc.

(A) Similar to last week, we have only documented our project using an online Scrum board called Trello. This board visually represents the progress of the team and includes three columns with notes in respective column ('to do', 'doing', 'done'). The reason why we use this tool is partly to keep track of the progress and illustrate each stage of work in the project, but also to visualize what to complete in each iteration. In short, the tool is used to visually represent the progress of the team. We also draw pictures of the different app screens on paper sheets, write reflections in text documents, and comment our code in the java files.

(B) In a future project, there are other options of documenting our work that we might use:

  - Use case, a way to document your code by showing how the app will act in a specific use scenario
  - Several UML (Unified Modeling Language) diagrams, such as interaction diagrams and class diagrams, would have provided an overview of the code and how different activities, methods and classes are connected.
  - Javadoc would be useful in order to better understand what the different methods and classes do.

(A→B) To get there, we need to include documentation in the tasks and in the acceptance criteria. We must put 'UML diagrams and Javadoc' as an item in the Definition of Done checklist and dedicate time to create them properly.

- how you use and update your documentation throughout the sprints

(A) During our work we have used Trello to set up tasks and user stories, and we have continuously updated this as we progressed. Google drive was used to write all reports, which were then uploaded to Github. Github was naturally also used to store the project and all code.

(B) In a future project, beside the tools described above, we would use UML diagrams to get an overview of our project and be able to see our progress. Use cases would allow us to demonstrate added functionality to other team members and product owners as well as test our code for a specific situation. All documentation would be updated regularly after completing each task.

(A→B) To get there, we need to create online UML diagrams by using a resource such as draw.io and each team members must contribute to the work by regularly adding their classes and methods to the diagrams.
The team should also come up with several use cases for the app. The use cases should include both intended and unintended ways to use app.

- how you ensure code quality, enforce coding standards, and application of scrum

(A) From the beginning, our group did not think about introducing a special coding standard, which has posed a problem for us as the project work has proceeded. The functionality has has become complicated to handle, since a lot of it is contained in the same classes.

Our team has used a few coding standards, such as indentation and logical naming, and we tried to keep our code short and neat where possible. We ensured code quality by reviewing each other's code to find any deviations from team coding standards. Code that did not meet the standards was refactored when possible.

Github was used for version control, which enables easy review of any changes. Changes to the code could be withdrawn if necessary. A lot of code was produced in group sessions, where most of the team could get together and solve coding problems (and review code), leading to a higher coding standard. Scrum was enforced partly by the scrum master, and partly by the will of the team. Everyone made an effort to learn and apply scrum to the development process.

(B) Ideally, to make a project optimal, the structure, rules and new work environment should be discussed in the early stages of process. Weekly stand-up meetings should be compulsory and hard deadlines should be set.

Future projects can be improved by better planning, which provides a structure and increases code quality. This can be achieved by dividing functionality and putting it into separate classes as well as commenting all the code, which makes it easier to comprehend all the functionality. If the team was given a chance to redo the project from scratch, from day one we would study the new development environment, discuss strategies and book compulsory feedback meetings.

(A→B) In order to make these improvements in future projects we have to actually apply the ideas mentioned above, and not just speak empty words. We have to take firm action to apply more structured planning, everyone in the team needs to make the necessary self-studies to learn and familiarize with the development environments early on - so that we can focus on the actual app and on applying Scrum most efficiently to make everything flow much faster and easier. In this way we can get much more done in less time; something which is beneficial to all parties of the Scrum cycle.

Application of Scrum

- the roles you have used within the team

(A) Daniel Cebe was our scrum master, he organized the weekly stand-up meetings, booked rooms and reminded everyone to show up.
The other five persons were team members.

The product owners are Volvo Car Corporation and Chalmers University of Technology. Also, in an Agile project there are stakeholders - in our case, the end users who have installed our app.

(B) In future projects there will probably be different product owners, and entirely new teams with different roles. Also the users which you target with an app will vary from project to project. While this particular project was a school project with focus on automotive; hence Volvo was an appropriate product owner, and the target users would be people of a certain age with cars, a future project might be in an entirely different category. Each team member might as well take on other roles, different from the ones they assumed in this project.

(A→B) In order for us to explore other roles in a project applying the Scrum approach of software development (and take part in other projects in general), we will have to actively seek to participate in future projects, whether these may be school-related, work-related, or something else.

- the agile practices you have used for the current sprint

(A) In the final sprint, we took the few user stories that were not completed, divided them into tasks and made our final product backlog (eg. completing the store feature and perfecting user experience and user interface). We planned our final sprint, and produced a sprint backlog. We executed the sprint in a group session where a few team members did pair programming. We tried to make the team cross-functional, but the different functions of team members were not well-defined and everyone did a bit of everything. We also did some pair programming, but not as much as we could have.

(B) In a future project, we might add other agile practices. The team could be truly cross-functional in a clearer way than it has been. Testing could be done more continuously. We could have exhaustive coding standards. We could do more pair programming, which helps the team members learn from each other and create a better code.

(A→B) To make the team genuinely cross-functional, we must assign distinct roles to each member. To make the testing more continuous, we need to include it in the group programming sessions. We should add more coding standards beyond the few that we have, e.g. ‘Never ignore caught exceptions’. Finally, we should schedule regular pair programming sessions more often.

- the sprint review (either in terms of outcome of the current week's exercise or meeting the product owner)

(A) The whole team has met the product owner during a feedback session in the Open Arena, where we briefly talked to Håkan and accounted for our latest sprint, but we did not show him the app or the tasks completed in the last sprint. We also reviewed the final sprint as a team, and concluded that we have completed all our user stories. The team agreed that all the sprint has been successful.

(B) In a future project, every meeting with the product owner should show the concrete results of the last sprint.

(A→B) To get there, we need to show to the product owner the changes on scrum board and the new features implemented by the completed tasks of the last sprint.

- best practices for using new tools and technologies (IDEs, version control, scrum boards etc.)

(A) All the tool that we have used during the previous sprints are functioning well. The AndroidCAR emulator in Android Studio is working properly and simPy was used to perform simulation. Using Github for version control feels natural for everyone, and all team members have worked in their own branch and made commits to the master-branch. Trello was used to formulate the remaining tasks and move them to completed when the work was finished. Communication was mostly done through messenger, but a Discord channel has also been used for voice group discussion. We kept our usage of these tools quite basic, since none of us are experts and the project wasn't huge. E.g. you could obviously utilize more complex branching techniques when using a tool like Git, something we didn't find necessary in this particular case.

(B) In future projects these tools and technologies will all play an important role, and the practices for utilizing them will most likely be refined and improve over time. For example, maybe you can utilize the hidden capabilities of modern, multi-functioning IDEs for more purposes, or at least in a more efficient way, in the future. Keeping regular meetings not just in person, but also in an online voice chat such as Discord might also be a beneficial practice for utilizing voice chats. Surely you can create more advanced and useful branching techniques when utilizing a tool like version control as well, especially useful when working on more complex projects in even bigger teams (possibly entirely online).
We also envision that in future, programmers who develop applications for a car will be using VR technology to simulate real-life use in a more natural way than AndroidCAR simulator is able to do. Thus, a developer can test real-life usage use without the need for an expensive car simulator or a real car.

(A→B) In order for us to do what was stated in (B), we have to continue participating in future projects utilizing these tools (which most probably do in some variation), and while doing so continue to explore and gain deeper knowledge and experience in making the best possible use of these tools and technologies.

- relation to literature and guest lectures (how do your reflections relate to what others have to say?)

(A) Our application of Scrum shows a lot of similarities to what has been said on the guest lecture by Per Söderstam. He discussed software project development in terms of complexity, planning, layering and testing. Complexity has indeed been hard to handle, but it was made possible by planning, formulating user stories and dividing user stories into small tasks to be done piecemeal (layering). During our sprint reviews we tested our functionality and adapted it when needed.

(B) In a future project, we could could improve our reflections by relating it not only to guest lectures but also to literature.

(A→B) To achieve this, we need to studying the literature about agile development and Scrum. There are a few well-known books we could read:

- “Agile Estimating and Planning” by Mike Cohn
- “User Stories Applied: For Agile Software Development” by Mike Cohn
- “Agile Software Development, Principles, Patterns, and Practices” by Robert C. Martin
- “The Art of Agile Development” by James Shore

This would give us a deeper understanding of the different concepts included in the agile and Scrum development, and our reflections would be more profound.