Final Report - Yusuf

Imad Alihodzic
Oskar Eggert
David Ferm
Gabriel Geraghty
Samuel Karlsson
Anders Olofsson
Johannes Rorsman

Github repo: https://github.com/iknek/DAT257-Team-Yusuf/tree/Code

Scrum board: https://trello.com/b/gRSHhKGJ/dat257

Customer Value and Scope

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

A:

We have been creating an application for the owner of a cafe in Stockholm. The aim has been to make it easy for the employees to store items that have been lost and found in the cafe and store information about the item, such as where it is stored, when it was found, as well as a description of the item. Value has been created for three parties. Firstly, for the owner of the cafe (the stakeholder), it increases the productivity of their employees and the level of customer service, as lost items are more quickly and correctly delivered back to customers. Secondly, value is created for the employees, as they can rely on a standardised method for finding customer's lost items, leading to less time being spent looking. Finally, the customers stand to benefit, as they are more likely to get their lost items back if they are located in the cafe.

The information about the lost items will only be stored on the device used to store it. As the device used to store lost and found items is located in the area where the employees meet the customers and receive questions about lost items, or found items, creating an application for this device generates the greatest amount of value. Since the device was an Android device, it was necessary for the application to be written in a programming language that is suitable for Android devices.

The stakeholder gave directions of which features were of greatest importance. The abilities to store items and then view them were fundamental and without them, the application would have no use. Other features that the stakeholder requested were:

- Information about when the item was added

- Location of the item
- Categories
- The ability to choose the order the items are displayed
- The ability to remove items
- The ability to edit item information
- Search functions, eg. search bar

The group also aimed to follow Scrum methodology as much as possible, and learn how to use it in practice. We also wanted to learn how to use Android Studio and Github in a proper manner.

B:

If we could start over again, we would have tried to include more stakeholders, and tried to create a more flexible application that could be used by other organisations as well. The application becomes useless if the organisation doesn't specifically have an Android device that is used by all employees when in contact with the customers. On the other hand we feel that we probably wouldn't have had time to create the improved application within the time frame of this course. We also feel that we could spend more time programming and less time planning and learning Scrum and Android Studio. Even though it was of much value to us, the stakeholder will not benefit from the increased knowledge since the time frame of the project is quite short.

A->B:

We should have reached out to more similar organisations in the beginning to find interest from other potential stakeholders as well. This would have helped us gain more knowledge about which features would be of most value, since it may be hard for the stakeholder to be sure too. We would also spend less time teaching each other about the parts that we manage since it takes up time and brings less value to the stakeholder than spending time programming.

The success criteria for the team in terms of what you want to achieve within the project (this can include the application, but also your learning outcomes, your teamwork, or your effort)

A:

The success criterias for the group were the following:

- Learn to work according to the Scrum principle

- Learn to use Github, Android Studio, Trello
- Work well as a group
- Create a functional app

Since the group's members were new to Scrum and the course offered knowledge about Scrum principles, the whole group considered the main objective to be to learn about it. Some members of the team were familiar, but not used to using Github, so for these members, learning how to use Github in an effective way was also important. Using Trello was quite new to some, so to get everybody on board and learn to use it was considered necessary. Most of the team members were not used to building Android applications, which made learning Android Studio another requirement for success.

Regarding the app, the purpose was to get a functional app that would satisfy the stakeholders expectations. However, we prioritised getting the group to work well together and to make sure everybody was involved and learning things as the project progressed.

To varying degrees, all of the success criteria were met.

B:

If the project were to be repeated, even more focus would have been spent on effectively applying the Scrum framework and following the practices and principles. We feel that even though we were focused on using the Scrum methodology, there was room for improvement.

We also feel that an app that could store information between devices would make it more flexible and therefore useful to other organisations. This would be a goal for us if we had the possibility to do this project again.

A→B:

One way to improve the learning of Scrum and also make better use of our time would be if we defined the roles in the team more strictly and over a longer timespan. Having an assigned scrum master for the entire duration of the course, as opposed to our weekly rotation, would ensure that this person becomes very confident in their role, allowing for a better implementation of Scrum.

The decision to create a more widely usable app with support for saving items between multiple devices would probably need to be taken early. A good way to improve the efficiency of the team would then be to let the people with experience in creating databases take care of that part of the development. While this would run the risk of hindering some team member's learning in certain aspects, it would largely still be beneficial to play to the strengths of the team.

Your user stories in terms of using a standard pattern, acceptance criteria, task breakdown and effort estimation and how this influenced the way you worked and created value

A:

We tried to use the template of *As X I want Y because Z*, but this was not always followed. Since some of the user stories were quite broad, using "because Z" made it easier to understand the reason the user stories should be implemented. When the user stories felt unclear, this also made what should be implemented more clear since the purpose of the user story was shown. Acceptance criteria were almost always created for each user story, and in those cases when they were not, it was because the user story was small and easily understood, making it unnecessary.

In the beginning, we felt that the user stories were often too big or too unclear, resulting in differing perceptions of when the user story was completed. We therefore started trying to quantify the degree of difficulty of the user stories by making estimations in the form of shirt sizes and fibonacci numbers. The team members who gave the highest and lowest sizes/numbers then explained their thought processes. This was very useful in the beginning as the team members were not familiar with each other's programming skills and knowledge. The user stories were created together in the planning meetings every Monday, after having communicated with the stakeholder.

Sometimes we found it difficult to create acceptance criteria for the UI-related user stories. This also contributed to broad, somewhat unclearly defined user stories. The lack of experience among some of the team members when it came to developing UI made the creation of these user stories a bit harder. However, the usage of Figma made it easier to understand other people's design ideas.

B:

We think it would be good to make even smaller user stories, as that would make it easier to work alone if someone wants to. It would also be good to have access to past size estimations of user stories' difficulty, as that would facilitate comparisons between weeks. Regarding the UI, it may have been nice to have a clear design in Figma all the time so that you are sure of how the app is supposed to look.

A→B:

We should have been more consistent when estimating each user story, and we should also have documented the results. That way we would have a more structured and standardised method of creating user stories. One idea would also be to create a small Figma mockup of the related views at each planning meeting, so that every team member has something other than the acceptance criteria to structure their solutions around. The potential benefit of this naturally depends on the effort put into making these diagrams.

Your acceptance tests, such as how they were performed, with whom, and which value they provided for you and the other stakeholders

A:

The application was mostly a front-end application and most of the user stories were related to UI. The back end and logic was tested by using the app and the team members that had

developed a feature tried using the feature and inserting different corner cases. The application had to be used to test the back end. The UI was harder to test, but when the Figma mockup was up to date, acceptance tests could be related to how the UI was compared to the Figma application. Another acceptance test consisted of checking if when a feature had been implemented, it worked on at least two different team members' devices. Later on in the project, we started to go through the features together via a shared screen on discord on the retrospective meeting. This helped bring up ideas about corner cases and led to some UI decisions being questioned.

The stakeholder was not involved in deciding on the acceptance tests related to UI, but was shown the Figma mockups regularly.

B:

The stakeholder should be more involved when it comes to defining what sort of acceptance tests are carried out. This would ensure that the focus remained on what the customer valued. The stakeholder should decide if testing or getting to use the application early should be prioritised. In a “real” project they would be paying for the work it takes to test an application.

We would also want to be able to test the application without having to use it. That way it would be more clear which of the user stories have already been tested and which cases were tested.

A → B:

We should have asked teachers or searched more for ways to test applications without having to use it. We also could invite the stakeholder to be more involved with acceptance testing, rather than just defining what features were to be implemented. More stakeholders with similar needs would also be useful so that at least one stakeholder could test the features considered critical by all the stakeholders more frequently.

The three KPIs you use for monitoring your progress and how you use them to improve your process

A:

Due to a misunderstanding of how we were supposed to use the KPIs we didn’t track our KPIs very precisely. Instead, we opted each week to reflect on the KPIs and whether or not they were at an acceptable level. This was first done verbally during the sprint review, and we would then reflect on the effectiveness of the KPIs in the Team Reflection. Due to this, we don’t have any data to share here. As a group, we felt that we were not entirely sure which KPIs to choose in the beginning, which would come to affect our work with them throughout the project.

- Burn-down rate

We considered burn-down rate to be a useful KPI since we could compare the amount of user stories completed between weeks, so that we got an understanding of our pace. Since the user stories were quite similar in size, this was helpful, however this KPI would probably have been even more interesting to look at if we also had estimated and documented the size of each user story. This way, the estimate would probably get even better.

- Carry-over

We chose to use sprint carry-over as a KPI because this would show how many user stories we would be able to finish each week, demonstrating that we were able to deliver on what we said during the planning of each sprint. An example of a use case for this KPI was during one sprint in particular, when we felt that we completed the user stories too early and didn't have anything left to do. We therefore intentionally added a little too many user stories so that we could check how many we really could complete in the next sprint.

- (Flow efficiency)

This KPI was selected at the start of the project without us fully understanding its purpose and use. This led to us swapping out this KPI for one we considered more useful.

- Hours worked

We would use this KPI to measure how much we as a team were working each week. We would discuss this during the sprint review to make sure we were working enough as a team. This KPI was sometimes combined with the burn-down rate and carry-over to sort out how many hours a user story takes in general. However, since the KPI was not recorded and was instead roughly estimated, it was mostly used to compare how much each person spent working. In that way, it proved useful when planning subsequent sprints, as it made it easier to ensure that every team member spent similar amounts of time on the course.

B:

We did not fully understand how to effectively use KPIs at the start of the project, which led to us not leveraging them to their full potential. If we were to redo the project we would be more diligent in recording the KPIs each week to be able to track and compare how we were doing each week, instead of only verbally communicating about them.

Tracking the KPIs more closely would have made it easier to follow the progress of each member, and also improved our planning, as estimates of each team member's work pace would be more accurate. We also would have liked to have introduced another KPI - *Team satisfaction*. We find that it would be relevant to measure the team's satisfaction over the project to know what a sustainable pace would look like for each team member. It would also be good to have the KPIs graphically presented in order to get a clearer understanding of the quantified KPIs.

A→B:

Discuss the choice of KPIs more in the beginning to make sure that they are useful and possible to track. Regarding the burn-down rate and carry-over, we think that an easy way to track the KPIs would be to just sum them up and document the numbers during the retrospective every Friday. Hours worked and team satisfaction could also be recorded in a table, perhaps in an Excel document.

Social Contract and Effort

Your social contract i.e., the rules that define how you work together as a team, how it influenced your work, and how it evolved during the project (this means, of course, you should create one in the first week and continuously update it when the need arrives) A:

Early on in the project, the group sat down and wrote a social contract together. This helped make sure that everyone was on the same page. Below is a list of the main points it included:

- Decisions will be made democratically within the group.
- Meetings will be held on Mondays and Thursdays
- People that come more than 10 minutes late must contribute 20kr to a fund for buying snacks.
- Everyone will put an equal amount of effort into the project. Communicate if you will not be able to do this.
- The focus will be on learning agile, and creating something that everyone can be proud of.
- If a conflict arises, speak up early, so we can solve it quickly.

There were some changes made to the contract along the way. Most importantly, we later added a definition of done. Any changes were made as people brought them up, and we did not regularly review the social contract.

Informally, we also deviated somewhat from the contract. We had our standup meetings on Wednesdays rather than Thursdays, and people arriving late opted to buy and bring biscuits or cakes themselves, instead of contributing to a fund.

The fact that we did not adhere rigidly to the contract was not a problem as there was quite a relaxed atmosphere in the group, as well as a general feeling that a strict, lengthy social contract was not entirely necessary.

B:

In future projects, it is of course not a guarantee that the group will work together and get along as well as this group did. It is a good general rule to stick to the social contract, and to be more proactive when it comes to continually updating it as new situations come up, as well as when changes are agreed upon. This is to ensure that everybody is on the same page, as well as to give everyone a chance to weigh in on any decisions that are made.

A→B:

In future projects, it would be best not to assume that the group will be able to work situations out without the aid of a social contract. Thus, it would be good to start writing the contract early on, and make sure to review it at least once every sprint. This would ensure that any deviations from the contract are brought up, potentially leading to changes.

The time you have spent on the course and how it relates to what you delivered (so keep track of your hours so you can describe the current situation)

A:

We all tried to spend half of our school week on the project. To do this we had sprint planning meetings followed by a group work session on Mondays. Then we would have 1-2 work sessions in smaller groups during the week, as well as some individual time spent on learning by ourselves. On Wednesdays we would have stand-up meetings and on Fridays we would have our sprint reviews. Over the course of the project we became better at scheduling work sessions ahead of time to avoid schedule conflicts as team members were available at different hours during the week.

There were some weeks when some people were not able to be quite as productive as the rest, usually due to other work getting in the way. This naturally led to slightly less work being done those weeks.

We did not clock our hours, but as described above, we generally spent a considerable amount of time working on the project each sprint. The work efficiency varied somewhat among the team members, due to the fact that some had more experience with software development than others, and thus did not need to spend as much time researching how to go about completing their user stories. Regardless, the progress that was made each sprint felt substantial and there was never any reason to suspect that anyone was not putting the time in, so we trusted each other.

B:

The evaluation of the time spent on the project in A was mainly based on subjective approximations. In future projects, it may be beneficial to be able to measure exact amounts of hours, in order to gain a complete understanding of the time it takes to complete the project's various parts. This would make it possible to analyse more exactly how productive and efficient the team is.

A→B:

We would ask each team member to clock the amount of hours they spend working on the project. This does not necessarily mean that each team member reports the exact times when they start and stop working, just that they track their total amount of hours.

Design decisions and product structure

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

A:

The UI as well as all the features that we chose to implement were decided on after consulting with our stakeholder. This included the order in which we were supposed to prioritise them. It was mainly in this way that we ensured that our decisions were always in support of customer value. The two biggest design decisions that were taken without the stakeholder's input were the use of Android Studio, as well as restructuring the app to use fragments.

We chose to make an Android app due to the fact that the stakeholder used an Android tablet in her business. We chose to use Android Studio integrated into IntelliJ for this, as it was a relatively simple environment that everyone in the group was at least somewhat comfortable working in, enabling us to achieve our goals. The project's short length made it important to try to limit the time spent on learning how to use the IDE as much as possible, so that we could quickly deliver something of value to the customer.

The focus for the first few sprints was to deliver a functional app with only the most essential features. This was because it was important to have a minimum viable product available. This priority meant that the initial architecture of the program was not as modular and optimised as it could have been, having quite a bare bones layout with several features on the same page. This ultimately made it necessary to spend time on restructuring the program somewhat later, spreading out the features among several views and classes, as well as accommodating for more to be added. Had we aimed to follow the design laid out in our Figma diagram from the start, we may have saved time overall.

However, some of this restructuring could be considered a bit of an overcorrection. Towards the end, we decided to change it so that the app's various views were fragments instead. This was in conjunction with implementing the navigation bar, which was supposed to appear on all views. We initially had the code for the navigation bar pasted into each and every class, and this worked, but was obviously not optimal since it meant a lot of code reuse. The change to fragments allowed us to have the code for the navigation bar in one place, and the customer value it provided was that it slightly improved the app's performance. That said, it is very possible that spending that time on other things might have been more beneficial for the customer.

We decided to forgo creating a database for storing the items, opting to store them locally. This was due to time constraints, as well as the fact that the stakeholder's business would only use one single Android tablet.

B:

For future projects, especially longer ones, having a clearer vision of what architecture the product will have would be preferable. Working towards the final architecture from the beginning may come at the cost of a minimum viable product being available slightly later, but ultimately it would save time and ensure that people in the team do not do work that must later be discarded.

Depending on the project's scope and the time available, it may also be good to consider more carefully what to prioritise. Perhaps a program does not need to be optimised to the furthest extent if it is quite simple.

A→B:

When working on the user stories, remember to add a "because" part in the description. This will promote more scrutiny when considering why certain things are to be implemented.

Have meetings in the beginning to decide on a definitive architecture. Attempt to provide a minimum viable product without sacrificing the overarching architecture of the program.

Which technical documentation you use and why (e.g. use cases, interaction diagrams, class diagrams, domain models or component diagrams, text documents)

A:

We have mostly worked with JavaDoc to document the code we were writing. This was to make sure that all the team members understood what each method did and was used for. To ensure that proper documentation was done, this was added as a requirement to the definition of done. Despite this, we were not always successful in writing documentation for every method, mostly due to forgetfulness. Because of the group's regular communication, it was still possible for everybody to understand the work that other team members had done, but this was a minor hindrance.

In addition to comments in the code we also created a diagram in Figma to illustrate our vision for the UI to the stakeholder. We created a first draft of our Figma, reviewed it as a group and then discussed it with our stakeholder. Following this, a newer version was brought forward, which then became the UI we worked towards. This version was not finalised until a few sprints in though, so for a while we were mainly working on the initial design.

B:

In future projects, especially more complex ones, it may be good to have more in-depth documentation relating to the functionality of the program. There were a few times during the project where there was some confusion as to how a user story was supposed to work, and this would have prevented that.

Having a finalised Figma diagram early on in the process would also have been good, as it would have made it clear to all team members how the layout was supposed to look, as well as how the features of the program were connected.

A→B:

Emphasise the importance of documenting each method, perhaps by creating a separate column on the scrum board for it. Sit down as a group early on and decide how the program's UI and features will look and function. Then create a Figma for the final project, and consider what documentation may be required.

How you use and update your documentation throughout the sprints

A:

Outside of technical documentation we have documented the process around the project too. We documented changes to our definition of done by updating the social contract, and took notes during meetings that were held. The team reflections we wrote documented some of the decisions taken during each sprint. These were all stored in a shared Google Drive folder.

We used Trello as a scrum board. This was mainly used to keep track of our various user stories. The main columns used were *Product Backlog, Sprint Backlog, In Progress, Testing & Documenting, Done*, but we also had a few others with things such as links to our repository, our definition of done, etc. The scrum board was continually updated with new user stories as it became necessary, and a large part of the sprint planning was deciding what stories that were to be placed in the sprint backlog.

We also had a Discord server for communication. This naturally involved both technical and non-technical information. This may not be considered documentation, but it provided some points of reference that we frequently returned to.

B:

For future projects, it may be of value to document each decision taken in greater detail. Some were documented in each weekly reflection, but these were not fully comprehensive when it came to describing everything that was done. Having a complete history of the decisions taken would be especially beneficial for new people joining the team, and would also enable a more informed analysis of the decision process.

A→B:

Write more notes during meetings. Perhaps someone in the group could take responsibility for doing so each time.

How you ensure code quality and enforce coding standards

A:

On the most basic level, we tested our program a lot, to make sure that the code had the desired outcome. The team considers the code in the final product to be serviceable, but with definite room for improvement.

There was a general understanding in the group that we were to write “good” code, however what that meant exactly was never fully discussed. We noticed a slight decline in code quality after a few sprints, in conjunction with the addition of several features. We thus started doing a code cleanup at the end of each sprint. This was typically done by one or two members of the team, and improved the modularity and general readability of the code.

Things such as coding patterns and rules were not decided upon. This was largely due to the use of Android Studio, as we concluded that it was not very intuitive to apply strict patterns such as model-view-controller when working within this framework.

B:

A clearer understanding of what constitutes good code would be good to have among the team members. This would likely improve the quality of new code written, as well as enable the people doing code cleanup to do a better job.

A→B:

Discuss early on in the project what the expectations are in terms of code quality. Make sure to have a designated time for code cleanup in the sprint. Adding a requirement for code quality in the definition of done may also be a solution, but this could risk slowing down the work.

Application of Scrum

The roles you have used within the team and their impact on your work

A:

During the project we have only really had three roles within the team: scrum master, product owner and the other team members. Due to the fact that the goal of this course was to learn to use Scrum, we decided to rotate the scrum master role so that there would be a new scrum master each week. As a result, six out of the seven group members got to try on the role of scrum master.

For the product owner role, Imad was selected, as he was frequently in contact with the stakeholder for whom we created the application. This allowed him to convey the stakeholder’s requirements to the team, even when the team was not in contact directly with the stakeholder. The impact of this was that it was clear who was communicating with the stakeholder, but it also meant that the rest of the team interacted less with the stakeholder.

The rest of the team members actively participated in Scrum meetings, but did not have specific roles. Instead, team members were typically divided into two groups of developers who worked on different user stories each week. The scrum master and product owner were included in these teams as well.

B:

To successfully apply Scrum it is important that everyone in the team knows what their job is and what is expected of them. It is good to have consistency from sprint to sprint, making slight improvements while not radically changing the process. Having the same scrum master throughout the entire project would allow for that person to grow in the position and more easily have an overview of the sprints. Switching the scrum master each week instead made it so it was always someone new and inexperienced on the job.

Having a designated product owner was good, but as a team we relied a bit too much on Imad to handle communication with the stakeholder. In a future project it would be beneficial if the entire team was more involved in communication with the stakeholder.

On the topic of other roles within the team, it might have been preferable to divide up certain tasks to create a more structured working environment. For instance, someone could have the responsibility for taking notes during meetings and someone else could make sure that timeboxed meetings did not go over time. Another potential implementation of roles would be to specialise more in the type of user stories team members worked on. That may be difficult for this project though, as team members had a somewhat similar background and therefore similar skill sets.

A→B:

To make the changes proposed it would be good to have a designated scrum master for the entire project. This person could then improve throughout the project and learn how to best work with the team. Scheduling demos with the stakeholder more often would allow the team to communicate more with the client and ask clarifying questions. More specific roles for other team members could also be created, to make sure it is clear who is responsible for what.

The agile practices you have used and their impact on your work

A:

Over the course of the project, we have used several tools and agile practices. We applied the Scrum framework of agile development and therefore had all the four main events of Scrum in some form. Sprint planning, sprint reviews and sprint retrospectives were a part of each sprint, and instead of a daily scrum meeting, we had a standup meeting mid-week on Wednesdays. Since we were not working full time on this project, a daily meeting was not considered necessary. Instead, we had sprint planning on Mondays, a scrum meeting on Wednesdays and then sprint review and retrospective on Fridays. This system worked well and allowed for adapting a user story or adding more user stories during the scrum meeting on Wednesdays, while a longer planning session at the start and the reflecting session at the end of sprints allowed for improvements from sprint to sprint.

Other agile practices we applied were the use of a scrum board and writing user stories to plan out the work of sprints. To do this, we used Trello to visualise our scrum board, making it easy both for team members and the stakeholder to see what we were working on in the project. This created a good structure for the project, allowing team members a good

overview to see the progress. We would each week pick user stories from our product backlog to put into the sprint backlog to decide what we focused on for the week and to make sure we had a reasonable workload.

To determine the workload for individual user stories we would try to estimate the user stories using planning poker. We started out trying to use the fibonacci numbers to do this, but as we felt that the estimations were arbitrary to us, we chose to change from numbers to t-shirt sizes. This was a good choice for us, as estimations felt more like an art than a science, something which the t-shirt sizes better reflected. The estimations would often spark a discussion on how much capacity a certain user story would require. In cases where the team decided a user story was too big we would split it up into smaller user stories. This was more common at the start of the project, but as the project progressed we realised the benefit of having a larger amount of small user stories instead of a few big ones.

B:

In the next agile project we work on we will be more methodical in our estimations. We could have been more scientific in our calculation of team capacity and velocity for each week which could have made us better at estimating. During this project we certainly did become better at estimating user stories for sprints, but we viewed it more as an opportunity to discuss the user stories more in detail to see if something needed to be changed. In hindsight, we could have been better at tracking our estimations and comparing them with reality to more consistently improve on estimations for the week and see less volatility in what we could ship each week.

For a project where only 20 hours per week is expected we thought that having meetings Monday, Wednesday and Friday worked well. It allowed for everyone to work throughout the week and check in every other day to see the progress. If the project instead is full-time we can see the benefit of daily scrums to make sure everyone is on the same page and working on the correct problems. Using a Trello board was also an efficient tool to create a scrum board and visualise the progress of the project.

A→B:

To create a more structured estimation process for the sprints, it would be good to methodically write down estimations during the sprint planning and then compare the estimations with reality at the end of the sprint at the review. Over multiple sprints this would allow the team to sense patterns in estimations and see where flaws in estimations were made.

The sprint review and how it relates to your scope and customer value (Did you have a PO, if yes, who?, if no, how did you carry out the review? Did the review result in a re-prioritisation of user stories? How did the reviews relate to your DoD? Did the feedback change your way of working?)

A:

Every sprint, we would have a meeting on the Friday where we did a sprint review and retrospective. This was when we would go over what had been during the sprint and what we could deliver. The product owner on our team would be the one responsible for making sure that the user stories we had completed during the sprint were aligned with what the stakeholder was looking for. After a demonstration for the team of what had been done during the week, we would look at the user stories to see if they had passed the acceptance criteria and the definition of done and could be moved from "Testing" to "Done" on the scrum board. After this we would have a discussion regarding the DoD to make sure that it still reflected our process. We changed the DoD a little during the first sprints as we decided to change our testing process. Below, you can see what the final DoD looked like. We started off with a requirement for unit tests, but after struggling with implementing them during the first week, we concluded that we could deliver a higher customer value by focusing on creating new features and instead allowing other team members to monkey test the application.

Definition of Done:

- Acceptance criteria must be met
- At least one team member should have reviewed the code
- Code should be merged without conflict
- To test UI features, we let an uninvolved team member try it out and either give it a pass or fail
- Every non-trivial method/class/variable should be documented (except getter and setters)
- User stories will be put in "testing" until the sprint review, where it will be checked and moved to done if working

As we had a single time set for both the review and the retrospective they tended to run into each other. We would not have separate meetings for both, which meant that we reflected on what we were delivering to the customer as well as how we were working at the same time. This could sometimes lead to somewhat unstructured reflections.

The sprint reviews functioned as a way to discuss the direction in which the project was headed. We discussed whether or not we were working on the right features according to the stakeholder. In some cases this would lead to reprioritisation of the backlog. After this discussion we would create a preliminary sprint backlog for the next sprint, which we would then have ready for the sprint planning phase of the following sprint. If something had gone wrong in the sprint, we would often reflect on how we could avoid that in the future. An example of this was that during the early sprints, it was brought up that having designated work sessions could allow for improved productivity. Before that, our work sessions had occurred in a more spontaneous fashion, which led to difficulties when team members had conflicting schedules. We did not, however, reflect as much on things when they were working. Instead of asking ourselves "Why is this working?", we viewed it as a sign that we did not need to change anything.

B:

In a future project we would try to have a better structure for the review. This would allow for the team to clearly define when we were conducting a review and a retrospective respectively. We would try to have the stakeholder present for more reviews if we were to do it again. This would allow for more direct feedback so that the product owner did not have to relay the information to the team. A good idea would be to dive a bit deeper into what was done in those occasions when we were able to deliver during sprints, to learn what was working. During this project, a lot of attention was instead directed to what did not work in the sprints.

A→B:

Timeboxing the sprint review and sprint retrospective to split them up could be effective. Inviting the stakeholder to more reviews would be helpful, though their schedule may not allow for it. Posing a question regarding positive outcomes during the sprint could also be beneficial.

Best practices for learning and using new tools and technologies (IDEs, version control, scrum boards etc.; do not only describe which tools you used but focus on how you developed the expertise to use them)

A:

New tools and technologies used during the project were Scrum and Android Studio. No one on the team had any substantial experience with either of them and we all learned together. For Scrum we used both the lectures, exercises, literature and practical experience. We took what we learned in the beginning and started applying it during the sprints. We would then use our reviews and meetings to discuss how to improve our application of Scrum. Opening up for discussions helped us teach each other during the sprints. For Android Studio, we let everyone learn a bit on their own, reading documentation and doing simpler hello world applications before the first sprint. We then divided the group into two teams of students, making sure there were people from both I and IT in each, and proceeded to work together. During a lot of the programming we would use pair programming, as we believed this would lead to a quicker learning process and higher code quality.

B:

Next project, it would be beneficial to have everyone be acquainted with the Scrum framework to allow for more time and energy on creating customer value. During this project, a lot of time went into learning how to apply Scrum. With the team members now being experienced with Scrum, it would be easier to consistently work on improving the process. Another benefit of this would be that more time could be spent on learning new technical skills or tools. An organised way of learning as a team could be beneficial too, as our method mainly consisted of individual learning with some attempts to share knowledge.

A→B:

Have team members with some experience or at least someone in the team who knows about Scrum and can teach the others.

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

A:

The lectures and literature were very important in shaping our understanding of the application of Scrum. Both the introductory lectures and especially *the Scrum Guide* by Schwaber and Sutherland (2020) laid a good foundation for the events of Scrum and how to effectively use the events to accomplish more during sprints. From the lectures we tried to embrace the concept of "slicing the cake vertically" for user stories. This idea was admittedly mostly present in the early weeks of the project, when the lectures were still fresh in the mind of the team members, and not as much later. Towards the latter part of the project, user stories instead mostly aimed to fix the last few things we needed to complete to end up with an MVP for the client.

B:

In this project, a lot of the information for Scrum and how to apply it came at the start of the project, and there was not much later on. The lack of real-life experience in the beginning made the principles presented feel very abstract. Looking back and reflecting on our experience with Scrum, it is now much easier to take in what is said in the literature. For this reason, we believe it would be beneficial for team members to actively revisit the literature throughout the project.

A→B:

We believe that reflecting on the literature more effectively over the course of the project would solve this issue. The scrum master could be given the responsibility to bring up relevant theory for the team to reflect on during reviews and retrospectives. Having a single scrum master for the project would naturally facilitate this.