

Final Report Team Naboo

Summary of weekly reflections

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

Climate Hero's scope was to make it easy to recycle correctly with the main target audience being kids. The idea of the design was an intuitive user interface with a point system which was our priority from the start. The team delivered an application which fulfilled the idea of making it easy to recycle correctly with an easy user interface. Since the app's main slogan is to "make waste zero" we are creating value for companies dealing with recycling since it increases the chances that items are actually recycled in the correct bin. On a global scale, we are creating value by evolving the youth to think more about the environment and therefore contributing to FN climate goal number 13 sustainability and goal number 11 sustainable communities and cities.

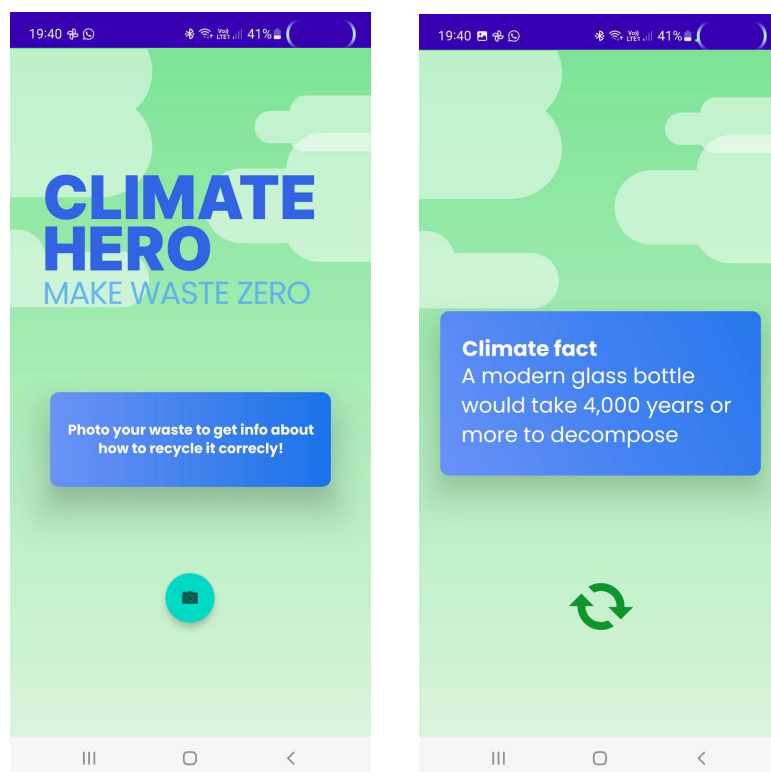


Figure 1 and 2: Screenshots from Android application.

B

Time limit was a factor that continuously influenced the project. Therefore the points system was omitted to make time for the main features that formed the application. Climate hero could be further developed to introduce a reversed points system. Since you don't want to reward the user to keep taking pictures of the same item because that would imply the user does not learn which bin the item belongs to. We want to reward the user for learning how to recycle. The introduction of a reversed points system would also mean we are introducing the ability to save user info since we would need a connection between a user and the point they got.

A → B

The feature of saving user info opens up new potential features that could bring even more value for recycling companies, for example seeing statistics of which items are being photographed the most. The reversed point system could also bring potential value towards learning how to recycle correctly since you don't want to end up with a negative score which further increases the value towards sustainable communities and cities.

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

We decided on the success criteria for the application to be really simple. We wanted to be able to take a photo of an item and the app would give a corresponding bin for that item. Also, the team wanted to get some experience developing a working android studio application. We had some challenges but we tackled those as a team and everyone was active in the project and had an idea of what was going on in the current sprint even though we had different tasks. We also set clear boundaries from the start about what the app should include. We therefore considered the effort for each sprint to be manageable and within 20 hours a week.

B

As previously mentioned the time limit made us prioritize between different tasks and therefore the reversed point system was omitted as an example. This shows however what a realistic scenario might look like in our later professional careers as software developers. We did however manage to fulfill our acceptance criteria of the application which we are happy with. However, we could improve our way of working agile.

A → B

For the next time we would develop an application in this type of scenario according to scrum we would choose a PO that was not on the development team. We sometimes lost the important contact and check-ins with the PO. Another area we could improve is just

developing what is specified in the current sprint and nothing else. We had some issues with the cloud database but we got it working in the middle of a sprint and we decided on a smart way of utilizing the cloud to fetch new addons to category bins to the local database but only when available. This was not specified in our tasks for the sprint and therefore we broke the way agile development should work.

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

To form our user stories we used the INVEST criteria. We consider that we formed our user stories well for the most part with a checklist of the acceptance criteria for the user stories. We divided the project into Epics MVP.a, MVP.b and MVP.c which essentially is for MVP.a it ties in with our acceptance criteria for the project, the ability to take a photo and get a result back. MVP.a is basically the foundation of the app and its value towards the product owner and companies working with recycling. MVP.b included nice to have features such as an app logo and climate facts. Features that bring value to the PO because it looks more professional with an app logo. MVP.c consists of features which would have provided additional value for the product owner as well as for different recycling companies which would have been able to see statistics and trends. We also split each user story into different tasks using the SMART criteria.

B

We managed to complete what we defined as MVP.a, and MVP.b but not MVP.c. This shows reasonable effort estimations for each user story but some of them were postponed to a later sprint. At the end of the project we were faced with two options introducing users but with no point system due to the time limit or adding random climate facts to the app. We considered the climate fact to bring more value to the application in the form of facts that might come as a surprise for the user teaching them something. However, in the long run the added functionality with user profiles and a point system would bring more value for both the user seeing the progression with points and recycling companies seeing recycling trends.

A → B

For the next time working with scrum to be more efficient and maybe have had the time to also complete MVP.c, we could have split some user stories a bit more. For the most part, we could finish our user stories in 3 or 4 days but some of the user stories took longer, indicating that we should have worked a little bit more with the S in the INVEST criteria. We consider the tasks we formed for each user story to be clear and well broken down fulfilling the SMART criteria. However, we could have worked a bit more with effort estimations using planning poker to involve the whole team in the effort estimations.

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

A

For a user story to be accepted the task/user story needed to pass automatic unit tests or manual tests and fulfil the acceptance criteria as well as be demonstrated for the PO by the development team. Not all tasks got tested with automatic unit tests but since our DoD stated that manual tests were sufficient to mark a task as done, manual tests were the most commonly used testing method. The tests were usually carried out by the team member assigned to the particular task. The purpose of our testing was to make sure that the new feature worked as intended, lived up to the acceptance criteria and did not break any previous feature of the app. When a user story was completed it was demonstrated to the PO during the Tuesday meeting to ensure that the promised customer value was delivered.

B

Since manual tests were widely used, most tasks only got tested once and could thus be vulnerable to faults caused by functionality added later that interfered with the old features. To ensure old code doesn't cause any trouble when adding new features creating more automatic unit tests would have been desirable. This could have in an earlier stage highlighted potential issues and corner cases that are hard to discover with manual testing. Furthermore, automatic unit tests also show how much of the code that get test coverage, which adds extra visibility about which code is properly tested and not.

A → B

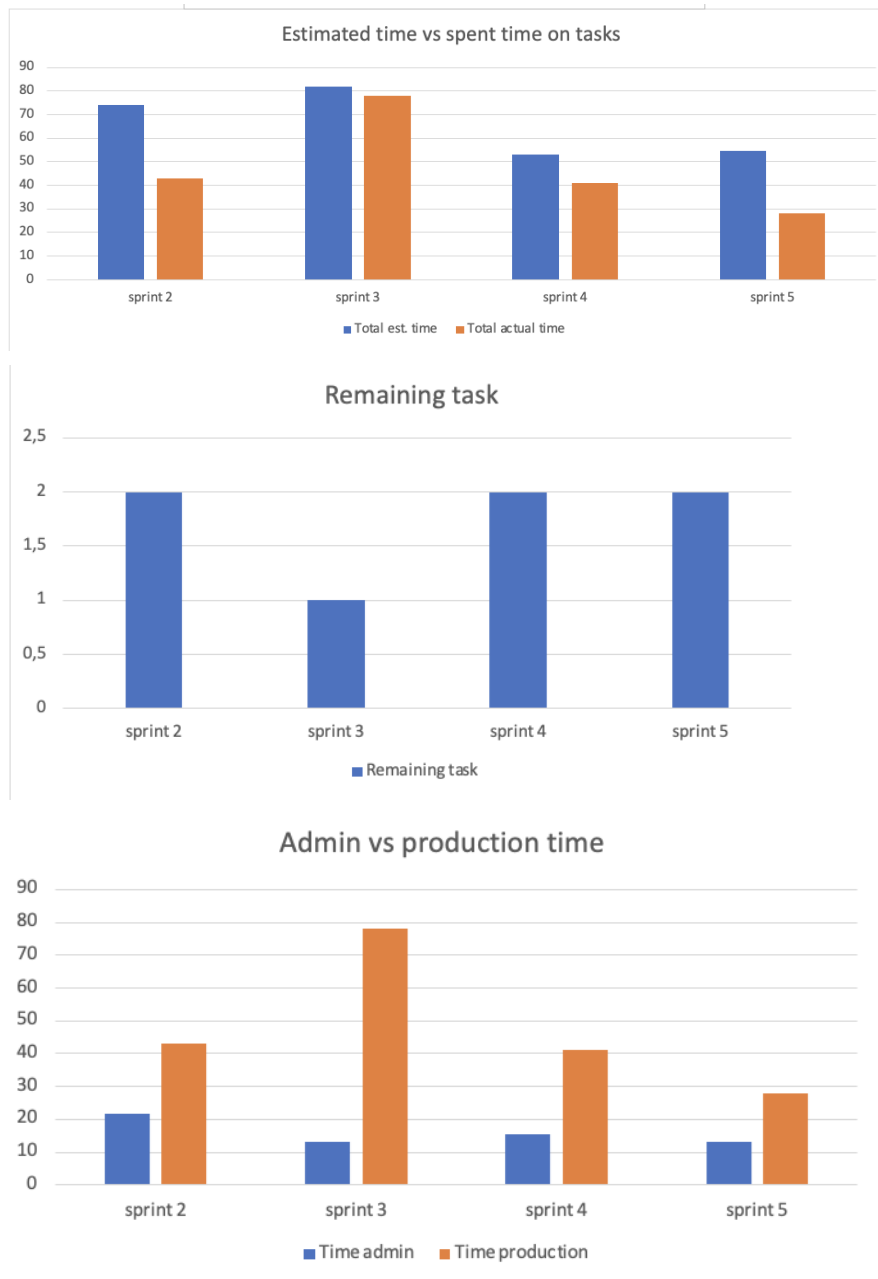
To make sure that automatic unit tests got more used, we could have modified our DoD to only allow us to mark a task as done when it had passed automatic unit tests. During manual testing, a demo for the rest of the development team could be suitable to ensure that the performed task is implemented as specified. It can sometimes be up to the developer to decide some of the specifications for a user story/task and it could thus be of value for the developer to argue for his implementation choices.

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

A

The KPIs used during the project was

- Estimated time vs time spent per task
Indicates how good we are at time estimating our tasks. Should ideally be equal to each other
- Remaining tasks after finished sprint
Indicates how good we are at picking the amount of tasks for each sprint. Should ideally be 0
- Total time spent, admin vs production
Indicates what we spend most time on, coding or meetings



Our KPIs throughout the project indicated that we continuously overestimate how much time a task will take. During the initial two sprints, our total estimated time was much higher than the two latter, the reason for this is probably because we did much of the “heavy” implementation, i.e. the database and API functionality during these sprints. This took much

time and as seen in sprint 3 was the one where we spent the most time and also was closest to the estimate.

The remaining tasks are ranging between 1 and 2 due to the fact that we used tasks for creating tests and extending the UML diagram. During sprint 3 we completed the UML for all implemented tasks and thus were considered as done.

The third KPI showing our distribution of time shows that we spend most of the time coding, which is as desired. The main admin time for each sprint is concentrated on Mondays and Tuesdays when we had our project meetings.

B

During this project, we have been keen to fill in the data needed for the KPIs after each sprint but we have not really evaluated them. As seen in the time estimation KPI we consistently overestimate the time a task would take, and ideally, we should have looked at this KPI to improve our time estimation abilities. If we had more thoroughly analysed the cause of the overestimations we could have fitted more user stories/tasks into each sprint since our social contract specified that we aimed to spend 20 hours per week on the project. The KPIs displaying the remaining tasks and time allocation were neither analysed much nor did not provide much value for analysing our performances. If we would have realised this during the project we might have replaced these KPIs with ones that would contribute more to seeing our actual performance.

A → B

Since the remaining tasks and time allocation KPIs did not provide much value these could have been replaced with some other metric. E.g. KPIs for the quantity of code written each sprint or how the workload was experienced each sprint would have been more useful and interesting to analyse. It was also a mistake to not analyse our KPIs while we had time since we took the time to fill them in to get the data. I think we just saw them as a thing “we had to do” and somewhat rushed through the KPIs in our meetings to get to more interesting topics such as new user stories and tasks. Having a follow-up on each KPI during our weekly sprint meeting would most likely also increase our attention and use of the KPIs. If we after one sprint saw that one KPI diverged in the wrong direction, this should have been highlighted and during the subsequent meeting be addressed again to see if we had a positive or negative trend.

Social Contract and Effort

Social Contract and definition of done

A

The social contract was made as an agreement with all our team members on how we would like to work together and with our project. We decided to meet in person once a week where

we finished our last sprint, wrote team reviews and planned our next sprint. Since we had this social contract from the beginning, the structure of our project and meetings were handled well, but improvements were made every week the more we proceeded into the project.

Together with our social contract we also defined our DoD (definition of done). One of the parts of our social contract was to follow the Swedish engineering codex, which wasn't considered very much since it refers to working in a company together with other engineers while we worked in a "simulated" environment defining POs and etc. on our own. Regarding the DoD, we added criteria for every user story and task so they had to be accepted before being added to the product.

B

If we were to start working on a new project and wanted to work agile, some improvements from our way of working with help from the social contract would be adding harder constraints to meetings and team reviews. This would maybe help in a situation where everyone is a stranger to one another, our group had the privilege that each member knew each other before working on this project.

A → B

A way to make a "harder" social contract, or in other words making social contracts that apply to a new group is to set up rules regarding meetings and effort in the team reviews, such as small "penalties" as in buying a cake for the team when the person in question missed a meeting without a reason. Another one would be to do a review after every sprint if everyone followed the parts when working on a task. In our group, we already expected it since we've known each other for almost two years, but in a new setting, it would help and improve the efforts.

Time/ Effort

A

The meetings with our group/ PO and writing team reflections were labelled as "admin time", while the rest of the time working with our sprint was labelled as "task time". Every week we estimated what all of our tasks would take in time. From the beginning, we decided to try to work up to 20h per person every sprint with either admin or task time, which would be a total of 100h of estimated time. The estimations of the time per task at the beginning of every sprint proved that it was a bit too much since we often managed to deliver a finished task in fewer hours than expected. This may be due to us being beginners in Android Studio, having to learn how to work with it along the way, but finding out that some parts aren't as hard as imagined.

B

An improvement would be to make time estimation more systematic e.g using the planning poker. Another thing is that our group was a small group only consisting of five people which was the minimum. If we could find two, or at least one, members to join our group, we could improve the time estimations and efforts in our project.

A → B

Our estimations in work hours were mainly dependent on our earlier experiences in programming, but as mentioned in part B using the planning poker cards could improve our estimations so they could be more accurate.

As mentioned we all knew each other before starting the project, but we could maybe use the course canvas side to find more participants.

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

A

After our initial application idea, we needed to decide what programming language and IDE we wanted to use. Since all team members had experience coding in Java, that became the language of choice. Being an object-oriented language, it would enable us to structure work in a manner that each individual member could work separately on a class/part of the application, and at a later stage integrate this with the whole project. It would enable us to both work modular and collaboratively. From previous projects, most of the members had used the MVC design pattern, so this was initially also chosen for this project.

In agreement with our PO, the application should run on an Android phone. For this purpose, we choose to use Android Studio as an IDE. Only one team member had previous experience working with Android Studio, so some time was spent setting up and learning the tool. We started with a design mockup that was proposed and accepted by the PO. During the whole project, we had several follow-up meetings with the PO to ensure that the different implementations were in line with customer value. Some decisions were however done without confirming the actual customer values, like the implementation of the internal database. This was an on-the-fly decision made by the team, to ensure that we were able to deliver a working MVP. This proved to be time well spent since it delivers value to end customers in the form of higher performance and fewer downloads.

B

Even though many of our design decisions were established early on, we could have spent more time being thorough in our planning stage. Many of our decisions and implementation grew organically over time. One example is our design pattern, in which we changed a few sprints from MVC to MVVC. This pattern works better when using Android Studio. Enforcing this pattern helped us develop faster and more structured, delivering results to our PO faster. While scrum did allow for these quick changes via feedback, unnecessary time was wasted refactoring.

A → B

Having even more tight communication with the PO and the whole team regarding what is expected in terms of customer value. Spending more time on User stories with clear acceptance criteria and then using these to create even more detailed tasks.

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

A

We used Jira throughout all of our sprints, in which we planned all our work and saved the development ideas. It was a great tool for keeping the process moving forward in an Agile manner.

The User stories were supposed to guide all work and act as documentation of acceptance criteria and functionality. The stories were later broken down into specific tasks with higher specificity. This worked well, but could perhaps be even more detailed to make it easier to be able to work on a task without many continued discussions with the rest of the team.

Our intentions were to create well-defined UML diagrams with all classes and connections, prior to the start of coding. These would show all dependencies and help in creating all necessary classes. Instead, we created the UML diagrams after the classes and functions were already done. The code was to be well commented on, so anyone could continue/improve/change any part of the application. This was never enforced though, so the code remains scarce of comments.

B

All classes and methods should be fully commented on. Tests should be designed according to well-specified requirements.

A → B

At the start of a project, it should be clearly stated what documentation is expected, in all parts of the project. It should be enforced and be part of the acceptance criteria for each task where it is applicable. In a small project like this one, where all members are involved with all parts, it's easy to think that documentation is unnecessary. But it's quite easy to see that as a project grows, both in size and in time, the need for good documentation is even more important. People can leave the project and new members need to be able to quickly get acquainted with the different interfaces and inner workings of the code.

The use and update of documentation, and code quality

A

As stated in the previous section, the actual documentation of the code was never really enforced. We did however actively work with our project documentation in Jira, and we continuously updated our KPIs.

In Jira, all User stories were defined and saved in the backlog. They were later divided into tasks and brought into weekly sprints according to PO and team prioritisation. At the weekly sprint review, all tasks were presented and decided upon, according to our DoD, whether they were considered complete or should be returned to the backlog for continuation in a coming sprint.

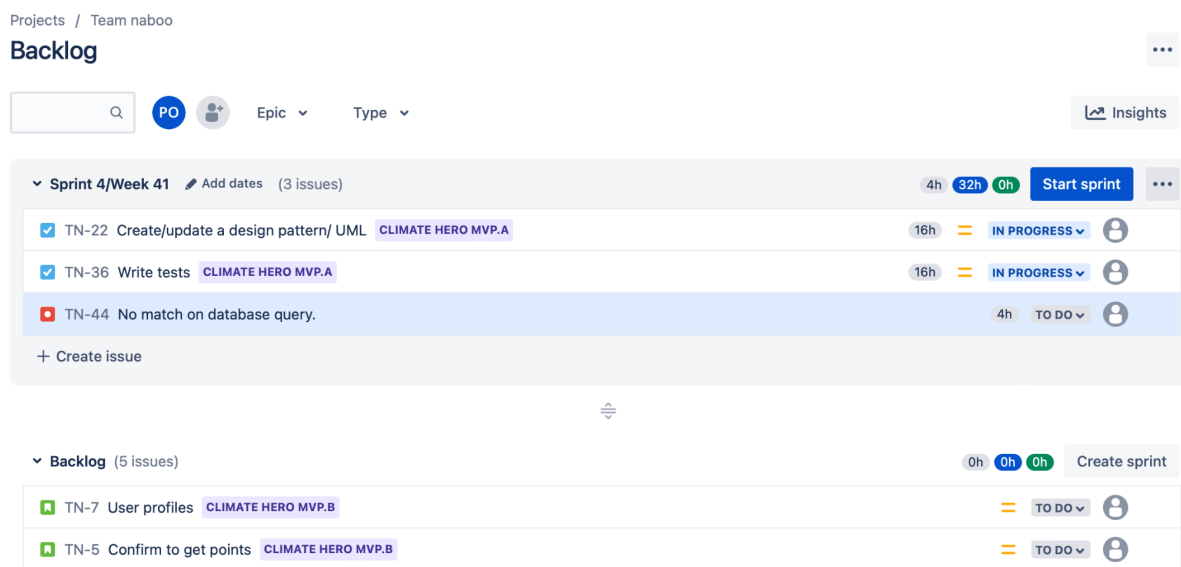


Figure 3: Backlog in Jira.

All KPIs were tracked and reflected upon to give us insights into future sprints.

Regarding code quality and coding standards, we did not specify how we would enforce good practice. Much coding was done in pairs, so corrections and help were enforced in real-time.

B

As far as our documentation in Jira and our KPIs, we followed and updated them more than sufficient for it to be of help to us throughout the project. With code quality and documentation, for future projects, there are many improvements to implement.

Higher focus on testing, each module should be implemented, if possible, with a test. The bigger the codebase becomes, the harder it will be to keep track of all changes and functionalities. Automated tests can quickly detect errors.

Well-documented/commented code. Greatly increase understanding of classes/methods logic when returning at a later time, both as the original creator or as a new programmer in the team.

A → B

All decisions regarding code structure should be addressed at the beginning of a project. Do there exist any industry standards that we can follow and adhere to?

Unit tests can be required to write alongside classes/methods. Finished work can be reviewed with another team member to verify that agreed-upon standards are followed. Prior to the new code making it into the official version, both integration and functional testing should be performed.

KPI's:

Sprint 4 - 4/10 - 11/10

Estimated time vs actual time spent, per task

Task name/ur	Estimated time	Time spent Robert	Time spent Johan	Time spent Pierre	Time spent Sebastian	Time spent Oscar	Done?	Total time spent on tasks
TN-40 Change the loading screen icon	2				0,5	4	1	4,5
TN-37 Local databasehandler	8	11			2	5	1	18
TN-36 Write tests	16			5,5			1	5,5
TN-38 Adding padding to sides of suggestion screen	1			1			1	1
TN-39 Remove top toolbar	2				3		1	3
TN-22 Create/update a design pattern/ UML	10							0
TN-41 Create an app icon and app splash screen	4			1			1	1
TN-42 Grant app permission to use camera	8		7				1	7
TN-43 Add recycle center to DB	2				1		1	1
Total:	9	53	11	7	7,5	6,5	9	41

Total time spent admin

Day	Time spent Robert	Time spent Johan	Time spent Pierre	Time spent Sebastian	Time spent Oscar	Total time spent on admin
Monday	1	1	1	1	1	5
Tuesday	2	2	2	2	2	10
Wednesday						0
Thursday						0
Friday						0
Saturday						0
Sunday				0,5		0,5
Total:	3	3	3,5	3	3	15,5

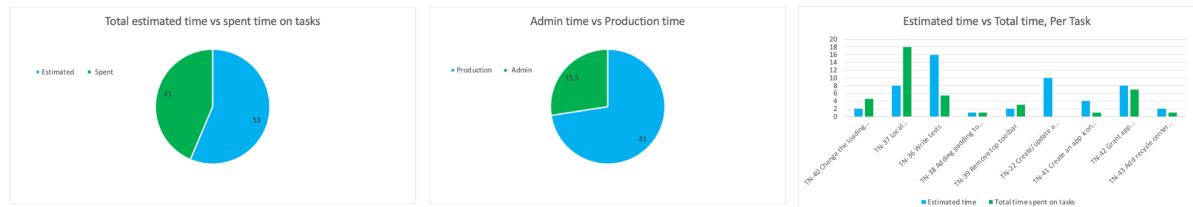


Figure 4: KPI diagrams and data for Sprint 4.

Application of Scrum

A

We spent week 36 organising the team, studying the fundamentals of scrum and planning the first sprint. During this week we:

- decided on a weekly meeting where we meet in person.
- set up social platforms where the team could communicate (Discord).
- created initial Definition-of-Done.
- set up version control (Gitlab).
- set up an agile task manager (Jira).
- defined our initial user stories and broke them down into tasks.
- appointed project owner and scrum master.

Our sprints were decided to span Tuesday-Monday since it suited the team best and also to have the best conditions to complete the project. Since the sprints were relatively short and daily scrum meetings were not advised, we spoke a lot about the highly important weekly meeting and that they must be prioritised by the whole team to have the best possible progress and also to discuss possible problems and risks that are identified.

We also discussed that the use of small user stories with supporting tasks that are connected to Epics will help us to define a common goal and get a good overview of the progress. Here Jira really helped us to organise the work and make the project more visual and lucid. We had some knowledge about the platform in the team but since the interface is very adaptable the team discussed and tested several set-ups. The stories were divided into several versions of MVP (a, b, c) based on priority, with a limited but carefully defined purpose.

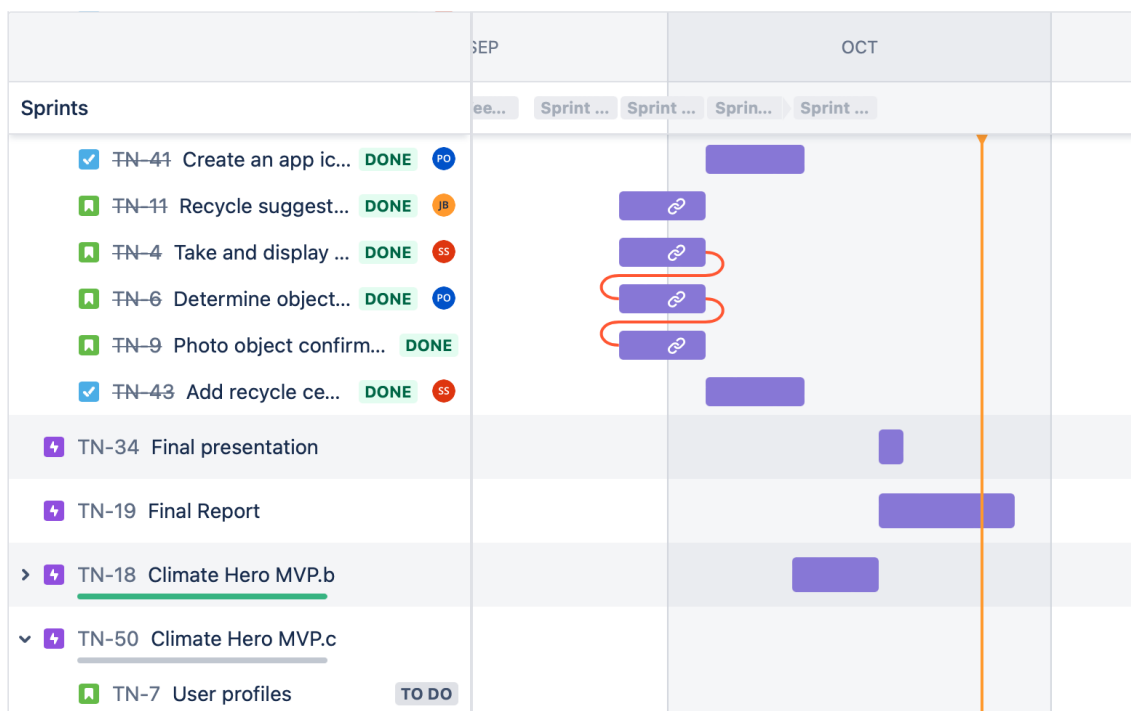









Figure 5: Roadmap in Jira.

We added Acceptance Criteria based on the scope and customer value of each user story that must be specified and later used as the Acceptance Test. This was to be evaluated in future weekly meetings.

Projects /  Team naboo /  TN-17 /  TN-4

Take and display photo

 Attach  Add a child issue  Link issue 

Description • Unsaved changes

As a user, I want to be able to press a button in the app to take a photo and display it since I want to recycle the photographed object

Acceptance criteria

As a user of the system I can:

1. press a button in the app to show the camera view and then to take a picture.
2. When I have the picture I want 3 options: Retake picture. Accept picture (for further analysis). Abort and return to previous menu.

Figure 6: Example of a User Story in Jira.

During the weekly meetings, we always started to reflect on the past sprint. Next, the PO (Oscar Larsson) and scrum master (Pierre Oskarsson) went through each user story in the current sprint to discuss the Acceptance Criteria with the team. If everybody was aligned that we passed the criteria and the Definition-of-Done, the task was declared finished. Finally, the sprint was completed, and possible unfinished tasks were moved to the next sprint and we made some more reflections on why.

During the reflections of our second sprint, we discussed that there should be a major change in the prioritisation of user stories. We had spent the previous sprint creating the basis for our Android application and some basic features, but we identified some technical implementations that would be essential to be able to deliver real customer value. This included communication with Google Vision Cloud API and the cloud database. If these would show to be a problem later, it could mean that the project scope would not be reached, thus they were stated as major risks. This resulted in us deciding together with the PO, that some of the user stories needed to be added and prioritized in the next sprint and that no more design, or other development work that didn't add directly to solving these major risks, was postponed.

B

The team got off to a really good start since fundamental decisions were made on platforms, tools, and technologies before the first sprint. There was also a clear vision of the collaboration. The team reflection proved to be a helpful tool which made us realise that we needed to re-arrange the tasks to minimise risks later in the project.

However, we know that we didn't adhere to the Definition-of-Done sufficiently. During the sprints, we often found some parts, like the implementation of tests, that a task didn't fulfil.

New features seemed more important at the time, and tasks such as creating tests and updating the UML did not get enough priority.

To highlight the need to do these tasks we converted this technical debt into new tasks but they got little attention and were often moved forward to the next sprint with the motivation that it will be completed later. Writing appropriate tests was part of the Definition-of Done, but since we created an independent testing task that lived through multiple sprints we marked the tasks as done and referred to the test implementation task.

A → B

We should not have allowed moving tasks to Done that did not adhere to the Definition-of-Done, especially regarding the necessity to create the required tests for each task. In hindsight, we should not have created our own task for test implementation but instead, reversed all tasks without the appropriate test from “Done” to “In progress”.

In the case of keeping the test task, we could have specified within the task which classes lacked tests and thus highlight which classes to write tests for. According to our KPIs, we often spent less time than the 20 hours specified in the Definition-of-Done due to overestimating the time needed for a task, so the team should have had the time available for creating appropriate tests.

Another possibility would be to update the Definition-of-Done and this is also something we could improve. The DoD should be reviewed and updated continuously to support a streamlined way of working. Thus we could have either removed the need for appropriate tests in the DoD and referred to the test- and UML tasks or been more strict about the DoD.

Conclusion

It is obvious that we should have adhered to the Definition-of-Done much more. Instead, we got a technical debt we needed to spend time on in the final sprints. This includes writing tests and documentation. For instance, creating an initial UML early in the project would have given us a ground strategy when starting to build the application.

However, by identifying the most complicated or uncertain tasks, we were able to eliminate the highest risks early in the project. This helped us to focus on the most important topics and minimise the waste of time that could have been realised if a big obstacle was discovered later in the project. What good is a nice-looking design if it doesn't work or important features are missing? In the end, customer value is the one factor that will keep a project going.