# Final report - Group Blue

## Customer Value and Scope

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

#### Step A: "what is"

In our project, we aimed to produce a web application (specifically for the Chrome browser) that compares alternative travelling methods such as taking the car, the bike, walking, public transport, and their respective $CO_2$ outputs. The goal was mainly to create value for commuters within Gothenburg who want to travel more eco-friendly. At first, we planned to implement a map on the site and show the routes. But after sprint 1, we decided to down prioritize the map because we could create greater value early on without it. We never got the time for it in our later sprints, but we delivered more than our expected Minimum Viable Product (MVP).

Prioritized features during the project:

- Place/address autocomplete input
- Time calculation
- Travel emissions
- Table to display data

#### Step B "what might or should be"

1: In retrospect, we could have picked a broader pick of location, e.g. the whole of Västra Götaland or even the whole of Sweden. Both of them would bring greater value to more commuters, our target audience.

2: One other thing that would be great to implement would be more transportation methods.

#### Step C: "feedback designed to reduce the gap"

1: Västra götaland would not have been that much more time consuming than Gothenburg, because Västtrafik and Openrouteservice (the two API's we used) both include Västra Götaland search results per default. Sweden as a whole however would have been much more time-consuming as we would have to implement much more public transportation API's (one for almost every county). But Openrouteservice would be able to provide the location needed for all of Sweden.

2: By adding “other travel methods” to our Project scope we could add new travel options along the way that feel relevant and possible to implement. For example traveling by wheelchair, which is provided by Openrouteservice, or electric scooter which are very similar to bikes.

# 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)

## Step A: "what is"

Our group's goals in this project were to achieve a working website with an appearance close to our mock-up. The site should be able to calculate and display time, cost and emissions from different methods of traveling.

In this project our learning outcomes has been that we can

- understand and code basic PHP code,
- use HTML and CSS to produce a website close to a mock-up,
- and work together as a team using Scrum.

In terms of teamwork, we had no clear goals or success criteria other than some points we wrote down in our social contract, where the following is relevant in this context:

- Everyone has an equal voice and valuable contribution.
- Be helpful if others need help with their tasks.
- Be courteous and respectful.
- Provide constructive criticism.
- Communicate before making changes to master.

If all this is done successfully, we think that we have succeeded in terms of working as a team.

## Step B-C:

In general, we are happy in terms of success criteria. However, if we would have to improve something, it would have been to start reflecting over our KPI's at an earlier stage.

# 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

## Step A: "what is"

Overall the user stories have been effective in providing a clear value proposition in the time frame of one sprint. They are easy to understand and evaluate in terms of priority, value created and how to begin working on them. Though the qualities of the user stories may vary both in terms of description, number of acceptance criteria and clarity for those acceptance criteria. Some user stories might have been big enough to possibly be categorized as an epic, or small and reliant enough on another user story to be an acceptance criteria of that user story. We used tags to mark effort estimation and what epic they might relate to for easier visibility on the scrum board.

Acceptance criteria has been used to narrow the specifications for a given user story. This has helped both in terms of having clear objectives for how to approach and complete the user story but also incurring a few discussions to change acceptance criteria to provide better value propositions. Likewise a few user stories regarding research like finding average CO2e values have even included suggestive criterias, aspects that need to be taken into consideration to either include or ignore.

When it comes to Task breakdown we haven’t really utilized Tasks more than initially. Seeing the smaller scope of our project and short sprints meant it was difficult to incorporate further effective task breakdown than the acceptance criterias in the already quite small user stories.

Our Effort estimations were based on a scale from 0.5-5 with 0.5 increments. In general effort estimations have helped to better organize and plan our work on the whole but also for any given week. Effort estimations made it easier to understand what we could deliver for any given week and organize our priorities partly around the amount of effort user stories required. We previously had increments of 1 but was deemed too difficult to access nuance when a user story was too easy for a 1 or perhaps more of a 3.5 than either 3 or 4. Our usage of effort estimations improved during the project but were still hard to pull off precisely.

## Step B "what might or should be"

1: More consistency in the quality of user stories to make them more clear and decrease the chance of something that ought to be an acceptance criteria becoming a user story.

2: More accurate effort estimations that better reflect user stories' difficulty for easier organization and reflection of our work.

3: Create more tasks to more effectively break down user stories in chunks that can be more easily managed and completed by the team and individual team members.

### Step C: "feedback designed to reduce the gap"

1: Provide time and responsibility to either one, for example PO, or more people to write user stories ensuring quality in terms of following standard patterns and being clear and easy to work with.

2: Record time spent, perhaps as a KPI, on individual user stories or sprints to better understand and define the relation between given effort points to the amount of effort required.

3: Providing time and effort to write tasks, either the team members involved in a particular user story or the PO before taking on a user story.

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

### Step A: "what is"

Our acceptance criteria:

- Changes in the code need group approval.
- The code should abide by our coding standard.
- New functionality should be integrated into the rest of the application.
- Approved changes should be pushed to the master branch.

We had two ways of performing our acceptance tests. The first was in the form of pull requests in GitHub, where the whole group had to review and approve before pushing to the master branch was allowed. This creates value in the form of better ensuring code quality since more people take a deeper look at the changes.

The second was a brief demonstration of the code and its functionality during a group meeting and getting verbal approval from the attending members. This way was added to prevent code from getting stuck with “waiting for approval” status if we were slow in reviewing pull requests. This creates value in the form of more frequent updates and making new additions available for other team members to work with sooner, at the cost of making our meetings run longer as well as the quality of the code reviews.

Both ways added value to the team in the form of everyone getting some understanding of the whole codebase and allowing anyone to offer feedback on the changes. With the whole team involved we increase our chances of discovering bugs and problems before they get integrated into the program.

Approval through demonstrations ended up being much more common (15 out of 19 merges) than the code reviews in github. Sometimes because we had been slow to review the code individually but other times because the pull request was added close to the

following meeting. This means very fresh pull requests got approval through a system intended to help approve old pull requests lagging behind. In those cases we lose the opportunity to gain value from a deeper look at the changes.

We have updated our acceptance criteria whenever we've made decisions that affect them, but we haven't actively discussed or considered new additions to them beyond that.

### Step B "what might or should be"

1: We could be reflecting on things that aren't part of our acceptance criteria, but maybe *should* be, because it's always good to be mindful of what can be improved.

2: It could be beneficial to get a better chance to review pull requests that were added close to an upcoming meeting as it would potentially give us value in terms of better code quality from the extra reviews.

3: We could have enforced or planned reviews to avoid having most pull requests get approval through meetings. Like the previous point it would give more opportunities to ensure code quality.

### Step C: "feedback designed to reduce the gap"

1: We could add a review of our acceptance criteria to our sprint review to see if we want to expand upon them for the next sprint.

2: Adding a cooldown period for pull requests where they can't be approved outside of github. The period could be relative to the size of the pull request as the review value grows with larger changes, and holding up small changes is less meaningful.

3: This would be easier to achieve with dedicated reviewers as planning gets harder the more people are involved. Planned code reviews could be assigned effort value like any other task and assigned to a suitable amount of members at the beginning of a sprint.

## The three KPIs (key performance indicators) you use for monitoring your progress and how you use them to improve your process

### Step A: "what is"

Our KPIs were:

1. The number of effort points selected each week.
2. The number of effort points in each list during the week
3. The number of effort points completed during the sprint vs how many were selected, (sprint burndown)

All of our KPIs were focused on effort points. Throughout the project we had trouble with effort point estimation, both the number of points a user story or task should have and how many we thought we would be able to complete in a week. In weeks 5, 6 and 7 we began the week with a number of points we thought were enough but later in the week we added more tasks. This was most noticeable in weeks 7 and 6. The problem with point estimation can be traced to the difficulty with determining a task's complexity. The fact that some tasks could be solved at the same time (which was not apparent to begin with) and that people sometimes took on more points than they had estimated in the beginning of the sprint. Many of us had not used PHP before and those of us that had some experience had not used it for this purpose before, therefore, those tasks were especially hard to estimate points for. This made it a bit difficult to use some of the KPIs to review our work progress.

We did not reflect much on the first KPI during the sprints as it is only of interest when comparing the number of points chosen in different weeks. Since this part of the project only lasted for four sprints, it was hard to get enough data to come to a conclusion on what a reasonable amount of points would be for each sprint. Our team was composed of seven members with different amounts of time available which shifted between sprints. This created an extra challenge when deciding how many points to take on during a sprint. Given more time we would probably be able to see a clearer pattern from the KPIs and more accurately estimate how many points we would be able to complete. We did, however, realise that our range of points was a bit too small and in later weeks added half points. Were we to continue we would probably switch to a different point system, for example using Fibonacci numbers.

We reflected more over the second KPI, mostly we looked at if tasks got stuck in a specific list on our scrum board for a longer time. We realised that during week 5 many tasks were stuck in the review (waiting for approval) list, waiting for a major restructure to be done. After that we tried to make the tasks more separated from each other and make sure that each person only had one user story at a time.

With the third KPI we ran into some of the same problems as with the first one, the difficulty with point estimation, but we could see if we managed to complete all the tasks we took on during a week. Week 4 to 6 we had some tasks that were not completed during the sprint, but week 7 we managed to complete all tasks.

## Step B "what might or should be"

In the best case scenario all of our KPIs should be useful during the sprints. The first KPI was not as useful as we thought it would be and it was not separated enough from the third KPI. Therefore, the first KPI should probably be replaced with another KPI that we could reflect on between the sprints.

We should also reflect more on the data during the project, for example it could have been a discussion point for the Sprint review.

### Step C: "feedback designed to reduce the gap"

The KPIs could be more well defined and motivated so their purpose is clear from the start. This would ensure that all KPIs would be of use during the project.

An example of a more useful KPI could be to measure how much time was spent on each user story to more effectively plan the sprint.

# 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)

### Step A: "what is"

Our social contract reflects our values with a focus on successful collaboration. This includes general guidelines such as being on time for meetings, how often we meet, to be respectful to one another and to provide constructive criticism. Our project specific guidelines are communicating before making changes to the master branch and that we resolve disagreements through voting. Everyone has an equal voice and valuable contribution.

Overall following the social contract worked well with some mistakes now and then but mostly individual incidents rather than systemic problems with team collaboration. In some cases of course a member of the group could be running a little late to a meeting or not be fully prepared which broke the agreements of the contract but it was never any larger issues that affected our work too much negatively.

The matters that we had agreed on were adapted by the group throughout all of the sprints. When we came to disagreements we used the method from the social contracts with using a poll where everyone had to take a stance and then the group accepted the result.

One of the agreements that proved to be more problematic was that our social contract stated that pull requests needed everybody's approval before getting merged with master. We had until the next group meeting to review (approve or request changes), without this we couldn’t merge to master. It was also in the contract that if this was not done by then a merge could be made at the next meeting, but then not everyone had an opportunity to check through the code and this could lead to less quality. So technically we never broke this agreement since many pull requests were merged at the meetings, but we intended the contract to encourage group members reviewing before meetings. We should have had it more specified in the contract.

The social contract was not updated on a weekly basis but instead when any issues would occur that we felt could benefit us, it was addressed in the contract henceforth. Only a few things were added to the contract after the two initial sprints.

## Step B "what might or should be"

1: We should review the social contract together as a group more often. By implementing this practice we could reflect on what went well on each topic in the contract and if something was an issue we could talk about it then and give attention to the problem right away.

2: We should also have more guidelines in the social contract, particularly for how to solve any larger disagreements or how to handle if one or more group members felt like another group member didn’t contribute to the project or perhaps didn’t complete their assigned tasks.

3: Another thing that can be improved upon in the social contract is to be more specific and precise. For example in our social contract, what exactly does the group define as being prepared for meetings or how should voting be conducted? Since this could mean different things for different group members it's important to add clarity and to be precise in the contract.

## Step C: "feedback designed to reduce the gap"

1: To reach this goal a group should discuss the agreements of the contract at the end of each sprint to evaluate what went well and what is needed to do better in the next sprint, or perhaps what needs to be removed from the contract moving forward. In the contract we had meetings planned and it would be possible to spend 15-20 minutes talking through the social contract on the weekly Friday or Monday meetings to conclude the past sprint. It could have been a distinguished part of our sprint review discussion.

2:

- To implement guidelines on how to react in more severe cases of disagreements between group members or how to handle a group member that was not contributing during the project's sprints is an important part that we did overlook. In the future we should keep in mind that it is important with guidelines for more extreme cases as well, even though we did not need this during our project this time. The best way to do this is at the beginning of a project before any conflicts could have started, that way if something happens there are already guidelines for the group on how to handle it.
- Another solution to this could be to reflect on resolutions within the group that did work, describe how the problem got solved and see if it is something that could be added to the social contract.

3: More effort should be spent in the beginning of a project when the social contract is being written. All topics should be specific and there should be no vague or ambiguous language used in the contract to avoid making topics open to interpretation. One solution that could

save time as well would be to find an existing social contract online to proceed from.

# 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)

## Step A: "what is"

We have allocated four hours per week for group meetings, including the TA meeting and the team reflections. The sprint backlog each week consists of tasks with effort values corresponding to our remaining estimated velocity/time.

In the past three weeks we have completed an average of 120 hours per week on planned tasks and activities, meaning an average of approximately 17 hours per person. Beyond that, we also spent time on our individual reflections and on code reviews.

We assign most tasks to groups of two to three people, making it easier to match the estimated time of tasks with the available time of our members since we don’t need to match each task with an individual member. As an example: we can have two stories of 4 points each even if only one member has 4 points available, because we can share the load across several members. In other words, teamwork increases the possible combinations of tasks we can add to the sprint backlog for a given week so we can focus on tasks that add the most value. However, we sometimes assign more people than really needed to the same task.

We’ve gradually become better at estimating the time needed for tasks, although we still frequently add more things to do during a sprint when our estimations have been off. Sometimes unexpected problems have occurred which drastically increased the time needed for a task. This could potentially extend its deadline to the next sprint. Our estimations for each member’s available time have been mostly accurate.

To help prevent our tasks having excess time assigned to them, we modified our effort scale of 1-5 (where one point equals four hours), to use increments of 0.5 instead of whole numbers.

## Step B "what might or should be"

1: We could have shorter meetings with less meandering discussions. Shorter meetings would free up time to do other things and could help reduce meeting fatigue as well.

2: It would be more time effective if we worked in smaller groups when tasks don’t need more people, as sometimes adding an extra person to a task doesn’t make enough of an impact on the time it takes to complete it.

### Step C: "feedback designed to reduce the gap"

1: A meeting protocol template could help our meetings be more effective as having an agenda would help focus our discussions, which potentially could save us some time. An added benefit is that it would be harder to forget to decide on something.

2: To encourage smaller groups, for smaller tasks in particular, we could simply ask ourselves if each new person we want to assign is really needed there or if their time could be better spent elsewhere.

# Design decisions and product structure

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

### Step A: "what is"

One of our APIs is openrouteservice as it had all the features we required, including all our desired traveling options, usage restrictions being beyond our use case and easy API to work with. Our other API is Västtrafik which we use to get more accurate time estimates for public transportation. We think that this supports customer value since commuters often travel using this mode of transportation.

As for the architecture of the product one of our choices was to make a website for our project. We did this instead of an app because we see it as a product that people are likely to not use very frequently, and therefore not want to download an app for.

The website is designed in a way that is user friendly. By this we mean that the user should have an easy time navigating the interface and for the website to be reactive, minimalistic and simple in scope. We think that this supports customer value as our user is more likely to enjoy their experience with a user friendly website. The website also makes use of symbols to make it easy to understand.

We also have many hover functions in the form of mouseover text to make the website more interactive and less cluttered. We think that this website is overall effective in providing what information is important and advanced enough to those who are extra interested in details, for example hover functions for calculations and added info.

### Step B "what might or should be"

1: An interactive map which shows the travel destination on the website. Here we could also use a GPS that can find the user’s current location to start from.

2: Reset function included so that the user can easily enter a new route.

### Step C: "feedback designed to reduce the gap"

1: Implement an interactive map in which the travel route could be displayed. This could include features which would provide value for the customer like for example having the ability to click on the map to mark the end destination.

2: We could implement a reset function so that once we were done with one travel route, we could easily try a new one. Currently we have to reload the page which is less user friendly.

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

### Step A: "what is"

Our technical documentation was kept in several different ways. It was partly done by creating sprint burndown charts, status burndown charts and effort points per sprint charts.

Mostly, our technical documentation was described in our README file. Here we describe the structure of the program with its main functions and their respective purposes in the program. We also include links to our developer documents including our Definition of Done, Social contract, Project scope, KPI:s and our current mockup which shows our website's interface .

### Step B "what might or should be"

1: Create a README file early in a project. Then, it will be easier to localize files and functions in the code without having to go in and look at them.

2: Add documentation for testing, which for example includes how different types were carried out.

3: Could use a tool to map the different functions such as a UML diagram.

### Step C: "feedback designed to reduce the gap"

1: We decided to create our README file towards the end of the project. We should do this sooner next time. We can make it easier for us by creating the README file in the beginning of the project, then updating it continuously as we create new functions.

2: Write test documents that include descriptions of how it was performed and the outcomes, both successful and potentially unexpected events.

3: Self explanatory.

# How you use and update your documentation throughout the sprints

## Step A: "what is"

In the first sprints we had a secretary assigned to keep notes during our meetings. These notes consisted of the most important things we discussed and decided on as a group, also if any document was updated. Anyone could look at the meeting notes if needed but it was rarely utilized as an asset. Towards the final couple of sprints the meeting notes were abandoned almost completely, and during all sprints it was more something that was done sporadically. The supervision meetings were however always kept note of and used for reflection afterwards.

The README file was created at the end during the final sprint and not used as a group asset during the sprints. We used comments in trello for continuous documentation between group members. By using Trello we documented who was working on which task, where in the process they were at the moment and relevant info for the tasks etc.

We also had documentation within the group in forms of a Definition of Done, Social Contract, Project Scope, KPI’s and a mockup for the website. The mockup was redefined halfway through the project when the map feature was deprioritized. The KPIs were mostly looked at and made towards the final couple of sprints. The social contract was the opposite, as described in another question above it was quite well done in the beginning and then not updated much. Our Definition of Done however was updated a bit through different sprints. One of the main changes was the group’s stance on testing requirements, if they should be more extensive by using written automated tests or if manual testing was enough. We ended up deciding that manual tests were enough for this project's needs.

## Step B "what might or should be"

1: Meeting notes should be kept more rigorously by someone responsible for that. By doing this the group can go back if needed and look at what has been decided. It would also cause less stress and confusion to members who could not attend a meeting if there were notes to look at afterwards.

2: A lot of the documentation could be utilized better by being revisited and reflected upon more often, perhaps on a weekly basis. By doing this issues can be solved faster, for example did everybody adhere to the Definition of Done, was there something from the social contract that did not work during this sprint etc.

## Step C: "feedback designed to reduce the gap"

1: It could be added to the social contract that all meetings should be documented. It is also important to have someone responsible, a secretary, like we had. This role could vary from week to week if needed so that it is not too much responsibility upon one person alone.

2: Since the group had designated time each week to do a sprint review the best solution would be to revisit the different types of documentation we used during that time. Then we

could review if the topics of the documents felt like they worked during the last sprint or not. This could also help with making a more structured and complete sprint review.

# How you ensure code quality and enforce coding standards

## Step A: "what is"

Our coding standard is part of our acceptance criteria and consists of:

- Commented functions
- Camel case
- Correct indentation
- Descriptive names for variables and functions
- Documentation at beginning of code documents
- Tested either manually or through automated tests

The idea is that code quality is ensured when changes in the code are checked by all team members before merging is done. However, we've struggled to find a balance between allowing thorough code reviews and enabling faster updates via merge through meetings.

We continued to feel that pull requests were getting stuck in "waiting for approval" each week, but looking back we see that the average time for a pull request to get merged is really just over one day. We go into more detail about pull requests and merging in meetings in our chapter about acceptance criteria.

Our coding standard is very brief which makes it easy to remember and adhere to, but it also means it permits different styles and inconsistencies so the code becomes harder to read as a result. Whenever we notice problems in that area we need to update our standard for the future and add tasks to fix past code.

## Step B "what might or should be"

1: A more comprehensive coding standard to reduce going back and fixing things when we notice gaps that need to be filled.

## Step C: "feedback designed to reduce the gap"

1: There are prepared standards and guides online we could use or incorporate parts from into our own. We could start by looking just at general guidelines or big topics like comments to not get overwhelmed by hundred pages long style guides.

# Application of Scrum

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

### Step A: "what is"

In general the team has worked collaboratively as a whole team rather than providing clear distinctive responsibilities for individual roles. While we had three roles in total they were perhaps not fully specified or utilized.

The Scrum Master, Johanna Schüldt, had responsibility regarding parts of our KPIs but other than that had no clearly defined role of what to do.

The Product Owner, Max Norén, had no clear responsibility. While the product owner has been more engaged in writing user stories, providing feedback and reviewing user stories waiting for approval it is not something clearly defined.

The testers, Irja Vuorela and Joel Persson, on the other hand, had clearly defined roles. As testers they would, upon need, make automated tests for a given function. But due to time constraints, time to make tests, and the fact that most of our functions were quite simplistic the need for automated tests over manual tests never arose.

On the whole our testers were underutilized and while we had a Scrum Master and a Product owner they were assumed to be mostly normal members of the team rather than having unique responsibilities. Instead anyone could hold a meeting, write user stories and reviews for approval processes were done as a team.

### Step B "what might or should be"

1: A more involved Scrum Master that might better take a leader role.

2: A more focused Product Owner with a clearer picture of what creates value and use that knowledge to create better user stories and provide better feedback during sprint and review.

3: Create roles where deemed necessary to improve processes and provide them with clear responsibilities and time to complete them. For example in this project someone who’d check DoD requirements could improve quality in approval processes more effectively than trying to have the entire team look at all user stories.

### Step C: "feedback designed to reduce the gap"

1, 2 and 3: Provide all members that have assigned roles time, clear responsibilities and what their role entails. Should scrum master lead meetings? Should the product owner have veto over approval? Etc.

3: Reassess during the project if certain processes would be better off with a role with someone taking responsibility.

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

## Step A: "what is"

In general the iterative development process of Agile has assisted in keeping up the momentum of continuous deliveries and weekly reflections make sure we keep in mind and improve what doesn't work.

The Scrum Board was made using Trello and, along with the product backlog, epics and user stories, has helped organize our work. The epics and user stories have been effective in dividing our work into manageable tasks. The product backlog has worked well in keeping track of our priorities and what needs to be done next, while the Scrum Board has been a great aggregate for all this as well as keeping track of our progression throughout the project and within sprints. Although there are some issues with the Scrum board not being updated consistently, especially the product backlog. Sometimes old irrelevant user stories have remained in the product backlog even after change of scope and sometimes user stories aren't commented on or moved when they should have been.

Working with sprints, in this case a week long, has helped keep the momentum of our deliveries as it forces us to break down our objectives into manageable user stories and only take on the amount of work we can deliver within that week.

Daily standups were spread out to our 3 weekly meetings rather than daily. Standups daily were deemed too time consuming in regard to our short one week sprints. Likewise, more time between standups meant members had more to report as they might spend time on other courses from day to day. The recurring meetings helped our team's collaborative setup, allowing for updates on our deliveries, suggesting ideas, asking for help and approving user stories waiting for approval. Though the dates for meetings are planned their structure and content are not, which means the quality of a meeting varies a lot and sometimes involves figuring out topics to cover.

Weekly reflections, both individually and as a team, meetings and sprint reviews, was a good method to keep improving our working methodology and deliveries. By forcing us to reflect and keep reflecting throughout the week we can more quickly assess bad practices. This along with our recurring meetings and shared communication channel, messaging through discord along with trello comments and github reviews, meant issues were addressed with relative speed.

## Step B "what might or should be"

1: More effective sprint reviews with a clearer structure of the aspects we should be reflecting on. We would like to assess our DoD, social contract and other important documents more regularly on a weekly basis.

2: More time effective Daily standups in the form of our weekly meetings that better covers the progress of our deliveries and what needs to be further accessed.

3: More up-to-date Product Backlog with finished user story and clear prioritization to make it more clear what work we have ahead of us.

4: More clear and consistent activity on the Scrum board to easier follow the progression of our deliveries.

### Step C: "feedback designed to reduce the gap"

1 and 2:

- Provide a team member or members, perhaps the Scrum master, to spend time organizing and planning sprint reviews, daily standup and other meetings. Either reducing the time spent on meetings or increasing their effectiveness.
- Have standard protocols for our recurring meetings, sprint review and daily standups, to make them more consistent and easier to plan and execute.

3: Have a team member or members, perhaps the Product owner, be responsible for managing and updating the product backlog.

4: Create guidelines for when to update user stories with comments and when to move the cards for improved consistency and clarity for what happens on the board.

## 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?)

### Step A: "what is"

We did sprint reviews on Monday mornings, at the very end of the sprint. Our discussion points were: What did we like about the sprint? What did we want to change? Some topics were brought up during these meetings, but most of the discussion took place in other meetings and in closer proximity to when problems and such were discovered. We have made changes to our way of working during the week, when we found it necessary and re-prioritization of user stories were mostly done during the planning of the sprint. Some changes we made were: adding a “Waiting for approval” list in Trello, deciding that all team members had to approve before new code was merged with the main branch and making sure that tasks were separated enough from each other to avoid conflicts.

### Step B "what might or should be"

The sprint review could be more useful to the group by letting it be the main discussion about improvements for next sprints. It could also be useful to document the sprint review and what was discussed and improved upon each week. This was done some weeks but not kept consistently.

### Step C: "feedback designed to reduce the gap"

For the sprint review itself to mean more it should probably be placed on the same day as the team reflection, since they overlapped quite a bit.

We could also have a larger predefined agenda with important discussion points. For example we could have gone through documents such as the social contract and the data from our KPIs and reflected on them.

## 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)

### Step A: "what is"

We used Trello for managing the scrum board. GitHub was used for code version control during the project. It was also used as our primary repository for all data such as reflections and code. Most members of the group used Visual Studio Code as their IDE. Openrouteservice, Västtrafik API, PHP, HTML, Javascript and CSS were used for the application. The PHPUnit testing framework for PHP was used a bit in the beginning of the project but was later abandoned when it was voted within the group for tests to be done manually instead. Almost all of the communication, meetings, etc have taken place in our Discord channel dedicated to the project.

Our applied learning method was for everyone to look up information about a certain tool on their own and try to get the hang of it. We then sat together in Discord to discuss our current progress and helped each other if anyone got stuck somewhere. At the beginning of the course we had a workshop where everyone helped each other get their working environment up and running. Everyone had different previous experiences with for example PHP. Some were completely new to the language while others had basic knowledge about it from previous projects. Our mentioned learning method solved this, those with previous knowledge helped those who were completely new with the language a bit extra in the beginning. This meant that they could get the hang of the basics necessary for the project at a good pace. We continuously shared things we found would help improve workflow for others in the group.

This method solved the important problem that the learning curve for new tools and technologies had to stay within the borders of the course. This meant that they could actually be utilized within the timespan of the course. By implementing this method everyone in the group could develop the expertise to use them in a reasonable amount of time that fit the length of the project.

### Step B "what might or should be"

1: Use a scrum board manager with more features aimed at software development teams, which Trello does not offer.

2. Workshops for learning new collaboration tools in general, for example: Git, Trello and Discord.

### Step C: "feedback designed to reduce the gap"

1: Looking up what other scrum board managers offer especially with focus on software development support. This would enable us to take advantage of certain functions oriented towards software development. For example, support for tracking associated User Story cards is easier, instead of having to color code related cards as we had in Trello.

2: We could also have internal workshops about tools that we have found and want to learn within the group. We could also introduce to the teachers and TAs of the course to either recommend offering workshops in the project groups or host large workshops for everyone in the course about for example Git branch handling basics. In this way you can start off with some basic knowledge right from the start which you will certainly use at least a few times during the project/course.

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

### Step A: "what is"

We have kept up communication with our supervisor during the entire project through Zoom meetings and a seperate Discord channel. Each week we discussed our progress so far and what our focus for the upcoming week was. We were given tips and feedback that we reflected upon and took into consideration when we continued working. For example, the flow of user stories in our Trello board had a tendency to come to a halt at the “Waiting for approval” stage. Much of the other feedback was the result of questions whether or not we had taken things into consideration that we had not thought of ourselves.

The team hasn't used external literature or videos to improve our work.

## Step B "what might or should be"

1: In future development we could focus more on different perspectives to help us gain insight in things we can improve with our work that we are not thinking of ourselves.

2: Have more active discussion with our supervisor and examiners through mail and Discord. Somewhat linked with the previous point, more discussion brings more perspectives.

3: Resort to literature and/or video to gain further knowledge about a certain topic.

## Step C: "feedback designed to reduce the gap"

1: Allow more time for scrum master and product owner to engage more in questions as “team leader” and “customer” and weigh their input as we have done with that of our assigned supervisor.

2: We could discuss and prepare points and topics more clearly in the group before meetings with the examiners and supervisor.

3: Assign either everyone or someone to find informative literature/videos on topics or methods that occur in our work, for example agile processes. These could then be reviewed and discussed together in the group.