Final Reflection

Group 13

2020

Anas Alkoutli
Carl-Magnus Wall
Erik Torstensson
Jakob Sikström
Julia Molin
Obada Al Khayat
Samuel Wallson

Customer Value and Scope

The chosen scope of the application:

The scope of this project was to create a tool to help citizens of Gothenburg to avoid parking tickets on residential parking spots. The stakeholders had noticed a gap in their already provided services in this area, which in addition to the fact that open data detailing both residential parking areas and their corresponding cleaning times were already being collected and provided openly, sparked the idea of creating this tool. Our stakeholders include a representative responsible for digital services in Gothenburg city as well as two representatives from *Parkering Göteborg*.

The scope was narrowed down to creating a web application including features for finding available residential parkings. Other features include notifying when a user's current parking spot becomes unavailable, to avoid parking tickets, and relaying information about parking restrictions. The scope also includes that all code will be published online, to inspire other developers working on similar projects in other cities. We also strived to set a good example of how to utilize open data in a simple and effective way.

Changes to scope during the process:

The scope of the project has changed during development, due to the inclusion of "Parkering Göteborg" as a stakeholder, as well as better satisfying current city needs. After week four of the project the scope changed from including all cleaning times for parking in Gothenburg city to specifying and focusing on residential parking. This led to different prioritization regarding features to implement, as well as requiring some rework of the code.

The initial scope of the project was to provide drivers in Gothenburg with an easy way to check when a certain parking spot is scheduled for cleaning in order to avoid a fine. The project was aided by open data detailing cleaning times for individual parking spots in Gothenburg.
This data is provided by Gothenburg City and is free for anyone to use. Our initial stakeholder was the development leader for digital services working within the municipality of Gothenburg city. The plan was to provide this service either through a mobile application or a web application. Based on the stakeholders' idea that feature specific mobile applications are unnecessary, and that users tend to not want to download them, we chose to go with the web application approach. Notable features, that were important to the stakeholder, were the option to add a notification when your selected street was being cleaned, in order to avoid fine, and a feature to suggest nearby parking.

Our initial stakeholder suggested the group to contact the provider of a popular parking application in Gothenburg, *Parkering Göteborg,* for a possible inclusion of our features into their existing application. *Parking Göteborg* were interested in our project and gladly joined the project as a second stakeholder. We thoroughly discussed our initial scope with them, but they already had a similar ongoing project. Instead, focus shifted to functionality

regarding residential parking information in the city. At this point in time *Parkering Göteborg* does not offer a solution where users can get detailed information regarding availability of residential parking spots, so we were advised to develop such a service. Our initial stakeholder was pleased with this new direction and the scope was therefore changed. Since residential parking users were not the target users for the existing app, the decision to make a stand alone application was made.

This resulted in the final scope of creating a stand alone web application focusing on the availability of residential parking. The main goal of this application is to make the parking situation for the citizens of Gothenburg a bit less complicated. We believe that by providing this service instead of the initially suggested one, we will be able to provide more value to the end users.

Near the end of our project we did some user research in terms of an online survey. This confirmed a lot of our assumptions about the problems with residential parkings. The majority of the respondents said that they would use such an application at least once or twice a week, where more than a third said they would use it every day. It also confirmed that users struggle to find parking spots and that it is common to get parking tickets due to forgetting about different parking restrictions. The survey seemed to confirm the value of all implemented functionality, but it did also highlight potential new functionality. However, the most important customer value was already implemented.

What would we do differently?

In order to avoid non-value adding work and to make sure that we make the right prioritization we would try to finalize the scope of the project earlier. We would make sure that we follow the stakeholders directions in order to make the best application possible.

If we were to start a new project or restart this same project, we would make sure to get the final scope set in place early by having more contact with the stakeholder. This includes having more meetings and better communication, but also asking more questions. To make sure we ask the right questions we would make sure we understand certain problems at a deeper level and we think that the experience we have accumulated during this project has given us a better understanding of what questions to ask.

To gain more user insight, we would have conducted the user survey at an earlier state. This could help us to focus on the right functionality from the beginning.

Success criteria:

We set out to learn and to deliver a great application to our stakeholders. In our social contract we defined these expectations:

> *“Lära oss Scrum och Agil utveckling. Bygga en fungerande demo. Lägga ca 20h i veckan inkl. Föreläsningar*”

Overall we achieved and followed our expectations. We feel satisfied with our application and made a moderate effort during the course.

In a new project we would perhaps expand further on our expectations, seeing as those we set up for this project were rather simple. By expanding on our expectations, or success criteria, it might help guide us in setting up the effort and, in extension, the scope of the project. This should probably be done early in the project.

Customer Value:

(Describe which features that contribute to the different values)

1. Value to users by avoiding parking tickets.

This is the main value provided to the user, pitched by our initial stakeholder from the start. The value to the user is in the form of monetary gain from avoiding to pay a penalty of about 800 SEK. This is the main epic user story and most of the features assist in creating this value to the user. Notable features for this is the calendar reminder for both Google Calendar and Outlook. This lets the user add a reminder in their preferred calendar which will then remind them to move their car and hopefully avoid getting a ticket.

2. Value to cleaning companies by easier cleaning due to fewer wrongly parked cars.

By reducing the amount of wrongly parked cars during cleaning times, the work performed by cleaners will be easier and more efficient. This will save the cleaning company both time and resources and in turn save them money. Furthermore, the streets will be cleaner which provides value to all citizens of the city.

3. Value to Gothenburg city by showcasing open data usage.

By utilizing the data provided by the city we show the stakeholders the potential value of collecting and giving it out to the public for free. This project might also inspire other municipalities to follow Gothenburg.

User Stories

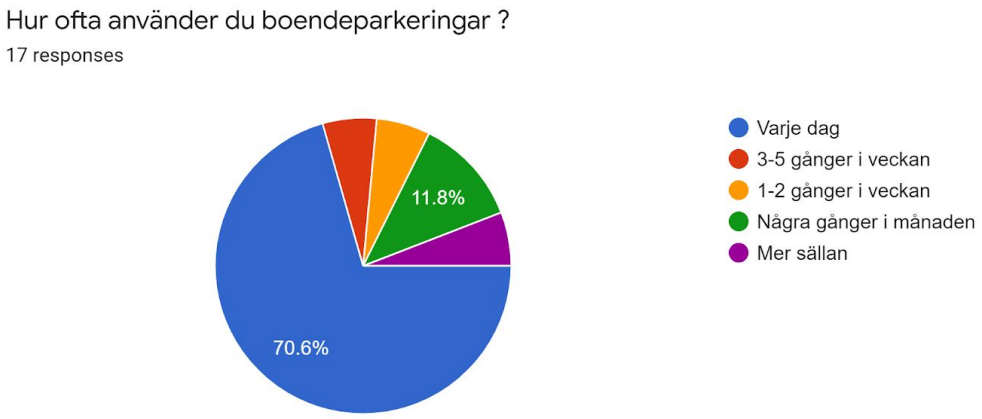
When developing user stories we did not use a particular pattern or system other than I.N.V.E.S.T. We made user stories based on what our stakeholder had told us and made them independent from each other.

We believe that our user stories were good enough. We understand that this is highly subjective and only based on our experiences with the user stories in question. However, we do think that the fact that we did not experience any issues regarding the size, or value scope of the stories, indicate successfully written user stories. There are of course problems

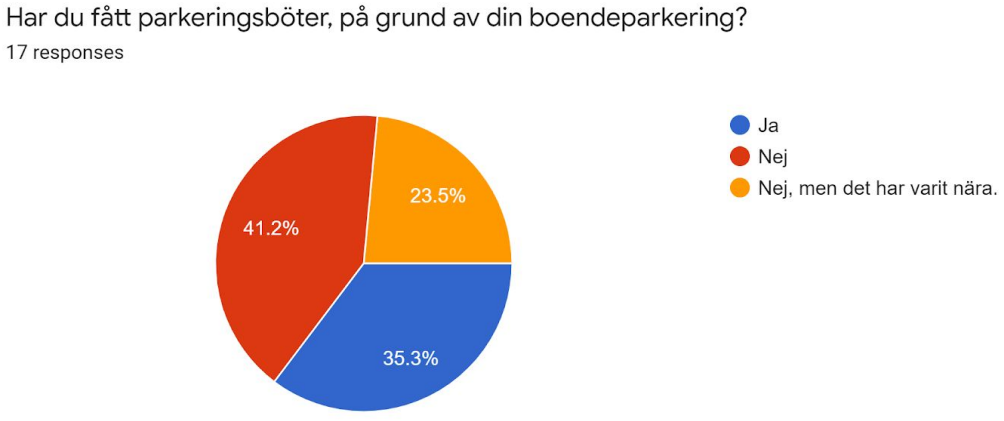
with the lack of a structure or pattern. An implementation of a good pattern will make sure that stories are of the right size, scope and not overlapping.

Verifying customer value through a survey

A survey was conducted as a request from our stakeholders in order to confirm customer value and user stories. The survey was distributed through two channels; shared on group members' Facebook pages and by placing flyers, with a QR code leading to the survey, on parked cars in residential parking spots in the city. We started by asking how often respondents use residential parking to ensure we reached the correct user group.



After ensuring we reached the right customer group we wanted to verify that parking tickets were an actual issue for this user group, which the results confirmed.



Lastly we wanted to know the reason for these tickets, to make sure we are combating the right issue. Answers brought up were; Missing cleaning times, forgot to move my car and missed night parking. All which could be helped with our implemented features.

This survey conclusively showed that there is a gap regarding information of residential parking in currently provided services. This confirmed the necessity of our project as well as the value it can bring to residents in Gothenburg city.

Acceptance Criteria

The acceptance criteria during the entire project was a weak point in structure. The acceptance criteria was not defined during the initial work or the first sprints. Instead, acceptance was based on subjective judgement during group meetings. Acceptance criteria during the later stages could be broken down into acceptance of tasks, merging conditions or stakeholder approval. Unfortunately, none of these followed a clear template except for merging conditions in the later sprints.

A criteria for determining if tasks were done was to create checklists in our scrum board on Trello. This was a way to break up tasks into micro tasks which would later be checked off to ensure that nothing was missed. One issue was that the person designated with completing a certain task also got to decide when one micro task was complete. This made checklists easy and comfortable to work with, but not very reliable. Another issue was that not all tasks got broken down into smaller parts due to different reasons. Some were small and descriptive enough that it would simply restate the original task, while others were not explored enough to know exactly what the tasks would entail, but only what we wanted to achieve with it. All and all, criteria for accepting tasks did not work very well, but it did not result in anything catastrophic due to group members working hard and being honest about when a task had been finished.

The acceptance criteria for merging new code into the main branch was at first agreed upon in a group meeting, where the person responsible would explain their work and remaining members would accept or deny the request. This would later prove to be inefficient and unstructured. Hence, we changed to a system where all members were picked as reviewers, and anyone who felt up for it could take a look at the new code, make sure they understood it and ultimately accept or deny the request to merge. This approach worked for a while but later resulted in many tasks being left unreviewed at the end of a sprint. Arguably, this was because the system lacked explicit personal responsibility. Finally, we worked out a system that worked; it was centered around a review template to make the structure of reviews easier for all members and ensure consistent code quality. This review template included the questions; “Does the code meet set acceptance criteria?”, ”Conflicts with existing code?”, “Is all code well commented?”, “Is it possible to extend the code without having to rewrite a lot of code?”, and “Do you understand the code?”. This combined with a minimum of two designated reviewers for each task made the reviewing and ensuring of the acceptance criteria being met an easy and well-functioning process.

The last criteria for a feature to be accepted was to showcase the changes to our stakeholders and have them give us their approval. This ensured that the task was implemented in a sufficient way, in order to provide value. This also gave an opportunity for our stakeholders to request changes and tweaks to the features. After gaining approval from the stakeholders, we were happy with the task and therefore marked it as completed.

This is one area where we as a group definitely could improve if we were to redo this assignment. First, we should have defined a clear acceptance criteria from the start, in order to reduce confusion and uphold consistent code quality during the entire project timeline. For

the micro tasks, we should have had a clear method for deciding when they were considered done. More objective acceptance criteria would have made this easier to see. This would ultimately have led the tasks to be more similar and the process as a whole to be easier to uphold for all tasks. Furthermore, it would have been great to have had the final version of the merge criteria implemented from the start of the very first sprint. This would reduce the amount of rework and backtracking needed later, as well as ensuring better quality and documentation of the code.

Key performance indicators

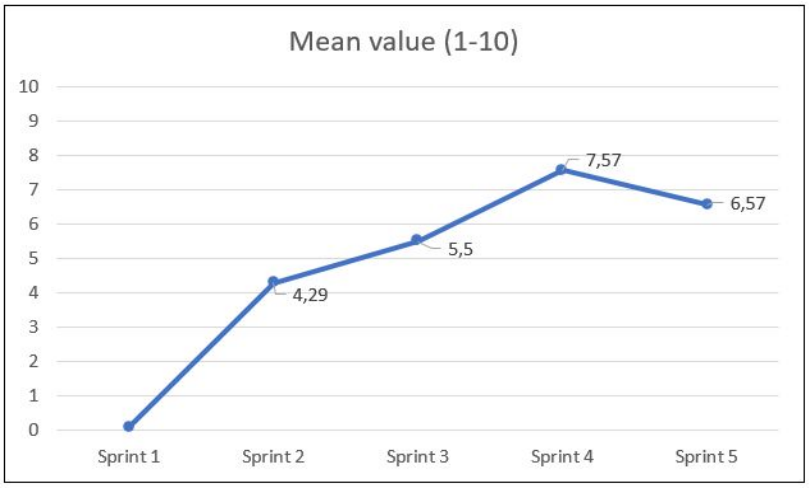
Our three key performance indicators were; velocity, stress and quality of code. These three KPIs were chosen because these give a good breadth of measuring different areas of the work process. The goal with these types of KPIs was to have indicators that would visualize a problem within the group in a short period of time.

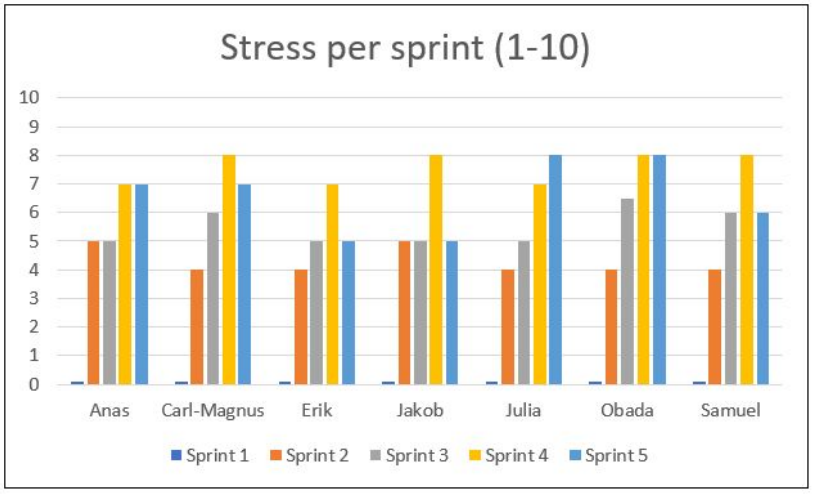
Process of choosing KPIs

As mentioned above, we wanted KPIs that would give instant feedback on how we performed and not something that would take a couple weeks before it becomes useful due to the project's short time span. It will appear later on during the evaluations of the KPIs that they were not implemented in the beginning of the project. This is due to our struggle to come up with useful KPIs. The first two, stress and velocity were decided early to be important aspects that should be measured somehow, but it took some time before the velocity was implemented due to the groups struggle to evaluate the effort that was required to finish a task. The last KPI was a bit harder to come up with. We discussed possible indicators during our supervision meetings and came to a conclusion that we wanted one about either code quality or value provided to our stakeholder. We discussed an optional indicator that would involve our stakeholder and their experience on our application, but due it being hard to quantify such an indicator we discarded that idea and instead had regular meetings with our stakeholders. Another KPI that was discussed was a test indicator, test-coverage/code coverage, but after research about others' implementations of the indicator we came to the conclusion that it would not be worth the time spent implementing and maintaining it. The third and final KPI ended up being a quality of code indicator, which is described in detail further down.

Stress

Our first KPI was a measurement of stress levels throughout the group. This KPI was evaluated by letting each member assign a stress value, ranging from one to ten, that they experienced during a certain sprint. This evaluation was performed during the sprint review, after the sprint finished and during weekly meetings. During the evaluation, group members had the opportunity to explain their stress value, which helped us decide on appropriate actions to take to maintain reasonable levels of stress. This KPI was mainly a way to make sure each member undertook an appropriate amount of work each week.





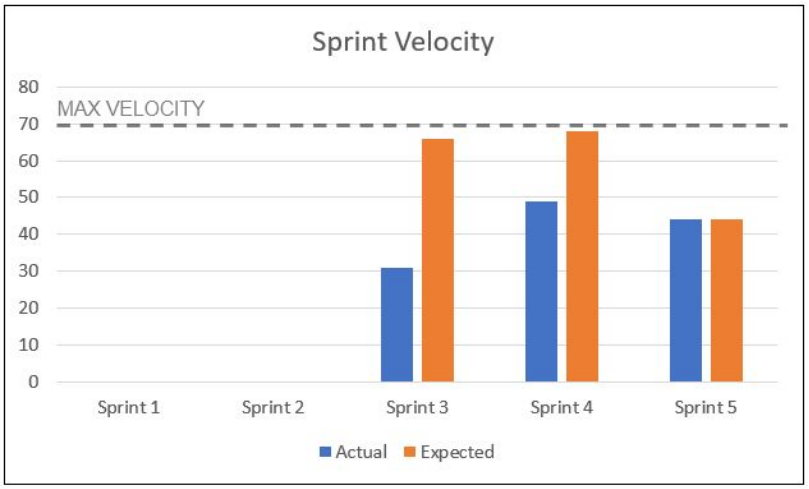
The stress levels varied significantly during the project. As a subjective judgement, we believe that a stress level of seven or below is satisfactory. As we can see from the graph, during some weeks we undertook slightly too many tasks, namely sprint 4. During other weeks certain members felt a stress level of five or lower, which would indicate that they could have taken on more responsibility during that sprint. It is also important to note that the presented stress levels included external factors, such as any parallel course many members were taking.

We were happy with this KPI and would use it similarly again in another project. One improvement that could be done would be to appreciate stress levels at the start of a sprint more formally and undertake an appropriate amount of work based on it. This could hopefully help us keep an overall stress level of 5-7 throughout an entire project.

Velocity

Our second KPI was designed to measure our effort. This was accomplished by deciding a maximum effort, in our case 70 effort points (10 for each member). In the planning of each sprint, each member had to estimate required effort for each and every task. Once every teammate had assigned an effort value, the actual effort was decided by calculating the average effort score. By looking at the effort scores, the sprint was then designed to include

as close to 70 effort points as possible. The KPI allowed us to come as close to our desired effort as possible and were meant to keep our sprints at a consistent effort level.



We started working with this KPI in our third sprint and managed to plan the sprint with an expected effort close to our maximum one. However, when the sprint was over we still had tasks that were scheduled for review, so we decided to not include these in the total effort.

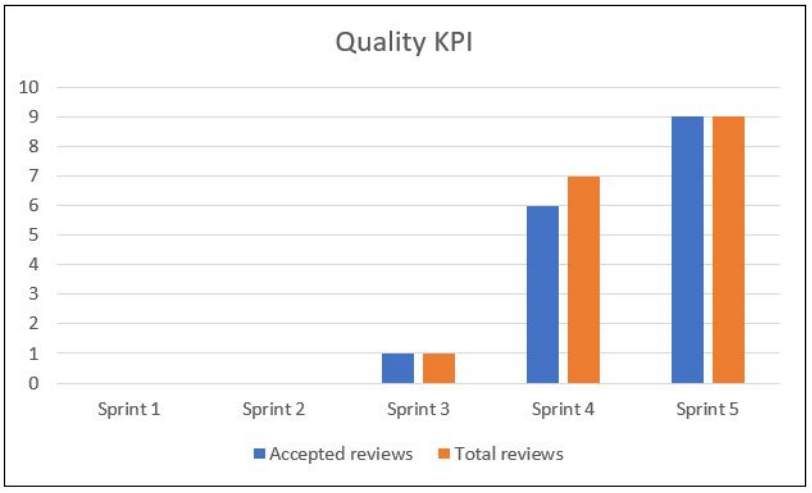
For the next sprint we also had tasks left in review and some tasks were not completed in time. We did however feel that our effort for the sprint was close to our prefered maximum.

For our last sprint, we intentionally set a low expected effort. This was done because we did not want to implement any new features, but instead focus on polishing on what we already had. The design of our sprint became more modular over time because of this, meaning that we discovered along the way what we wanted to fix or what we had time for. Overall, this meant that the actual effort effectively surpassed the expected one, but since we did not have all tasks from the beginning not all tasks were estimated in effort.

Ultimately, we were happy with this KPI as it helped us estimate our sprint effort and scope. We did not know in the beginning if we had the skill to accurately estimate effort. While some tasks seemed to be either too high or too low in estimation, we still achieved sprints with roughly the correct amount of effort. If we were to make a new project we would use this KPI again, and with our newfound knowledge in scrum and agile programming we think that the KPI would be even more useful.

Quality of code

The aim of our third KPI was to measure the quality of our committed code. We chose to measure this as “how many pull requests reached the acceptance criteria?”. We would most likely have changed this one if we would have continued the project further. Once we had specified our acceptance criterias, the review process started going more smoothly, which increased the overall code quality. However, due to stricter requirements, the number of rejected pull requests increased. So our KPI, as it is defined at now, could be a bit misleading.



The quality KPI shows significant inconsistency in the project workflow. However, this is not quite representative of reality though. Sprint 3 only had one merge because the system for choosing reviewers was not yet finalized nor working as expected. Sprint three had several pull requests to GitHub, but only one of them got reviewed before the end of the sprint. These requests all got accepted later, but were not counted towards sprint 3 or 4, meaning that several reviews did not appear on this graph.

This KPI relies heavily on other areas, namely acceptance criteria and the two designated reviewers. Because of this, the evaluation of this KPI was not easy and it suffered since the other areas were not yet finalized. We believe that this KPI could work well and we do not see any direct changes that we would make to its final structure. Instead, if we were to redo this project it would be better to make sure we finalize surrounding and affecting parameters before we start using this KPI to evaluate our code quality. This, along with starting earlier to get more samples to compare and more time to improve, would be very beneficial.

Social Contract and Effort

Our social contract worked well. We did not have any major issues or conflicts, and thus we barely discussed the contract after it was initiated. After a few sprints, we decided to have a Scrum master role which was rotation based. At the start of each following sprint we would randomly select a group member to act as Scrum master. This, along with a small part detailing the conduction of code reviews, were the only changes to our contract during the entire project. To ensure that a social contract is utilized efficiently in the future, where the collaboration might not be ideal, it could be a good idea to reflect and discuss the contract on a regular basis. One solution to this could be to include it as a topic in each sprint review.

A reason behind the lacking usage of the social contract might have been due to the contract not being thorough enough. The only actions set to be performed in case of a repeated wrongdoing were that the conflict would first be brought up internally and if that did not solve the problem, a supervisor would be contacted. In a dream scenario the social contract should be designed to represent the group’s values, behaviors and social norms, and therefore be something that each group member has in the back of their mind throughout the project. Creating an environment that hopefully everyone feels comfortable in and is

productive in. Creating these types of contracts would probably take a lot of time, but might have been worth the effort in a bigger project.

During the course of the project we did not keep track of hours spent. We felt that we could keep track of our efforts more effectively using the stress and effort KPI as they, as previously discussed, made sure the scope of each sprint was of the right size for the team. In our opinion this system worked out great and we did not feel at any time that we were missing a measurement of total hours spent.

If we were to continue the project or start it over again we would want some sort of measurement of the effort (or time) we spent on the project in relation to what is delivered. While we are of the opinion that our present system will work great as a measurement, implementing our total amount of hours might give us more insight in how much effort we actually put in during each sprint. Therefore, we would try using our hours spent working as a measurement in combination with our present system and later evaluate it. If we would feel that we get more utility from the expanded system we would opt for it for the rest of the project. If it would not satisfy our needs, we would be happy to keep using the present system.

Design Decisions and Product Structure

Several decisions, regarding our approach to deliver value to our stakeholders and provide a product that meet the expectations, have been made throughout the project. In this chapter we will discuss these decisions and how they have affected our work.

Design decisions and APIs

We decided early on in the project to use the Google Calendar API. The idea originated from our stakeholder, who expressed that the application should alert users using some sort of notification system. We discussed the idea thoroughly with our stakeholder, and asked if the notification should be sent from the application itself or, for example, from an already available calendar application. Our stakeholder recommended us to take the calendar approach due to the fact that many users already use digital calendars on a daily basis.

From this standpoint we asked ourselves which API to choose and finally decided on the Google Calendar API. We had heard that it was relatively easy to use, which would allow us to allocate more time to creating customer value. The implementation worked smoothly and the API was indeed easy to work with. Later on, we decided to also implement the Microsoft Graph API to make sure that our service is available to as many people as possible. Seeing as the two most common calendar applications are the ones provided via Google and Outlook, these were the obvious choices.

During our meetings with the stakeholder, about one third through the project, our scope changed to also include an interactive map. Our positive experience with the Google Calendar API along with our familiarity with Google Maps led us to work with the Google

Maps API for this purpose. The API had great documentation and many features that we felt we needed for our project. The API is widely used, which means that there are a lot of examples and forum threads on the internet, which all were useful for us during the project.

We feel that our tight cooperation with our stakeholder really helped us guide our choices of APIs through highly specified features and scope. This meant that the customer value always was in focus and we could be sure that we put out our effort toward the right things.

The chosen APIs would not be changed if we were to redo or expand on this project as we have yet to find any better solutions. We would, however, make sure to ask our stakeholders earlier on in the project and confirm which features are the most crucial, so that the implementation of all APIs can be done from the beginning. While it was not a problem this time, adding APIs over time might create problems in functionality or cause major rewrites that take time away from value creation.

If the project was to be expanded upon we would implement explicit design patterns. We did some file organizing to make sure the code was easier to expand, but since the application is supposed to be open source it would have been a lot better if it was designed according to one or several patterns. This would ensure that other developers have an easy time to quickly grasp the structure of our code and use it for their own projects.

Technical documentation

Our approach to documentation:

The technical documentation used in the project was not overwhelming. In the beginning it was up to the author of the code to comment and decide when it was sufficiently documented. Later on, after this had resulted in varied quality of comments, we implemented our new acceptance criteria, which included comment quality. During the reviews we had a criteria that a comment had to describe a function in a way that the reviewer could understand it to a certain extent. Our take on comment quality was to let the code reviewers determine if there were well-formulated comments alongside the code, as well if these comments were sufficient enough to understand the code function. These comments were formed after our guidelines. The guidelines were not too strict; the requirements to fulfill the guidelines were that the comments had to clearly describe how a function worked, what input it expected and make sure an outsider would understand it. Even though we felt like the code quality throughout the project was decent, an improvement in this department would definitely not have hurt.

What would we do different:

To make sure that the code is easy to understand and easy to expand upon by other developers, better code standards and documentation is probably needed. Since the possibility of expanding the code is something the stakeholder has asked for, this area is something we would focus on if we were to to further develop the application. Furthermore,

we as programmers would benefit from a more structured and documented code. We did not spend that much time figuring out how other authors' code worked, and we can clearly see how this could become an issue once the program becomes more complex. This did not become a problem in this project, mainly due to us extending our previous individually written code and not getting too involved in the features implemented by other members. This could become an issue in and of itself, since being as flexible as possible is something to strive for.

How would we do it?

By utilizing something like Javadoc we could have achieved a more comprehensive code by using a standardized comment structure as well as a clearer documentation of functions. However, we would always try to make sure to uphold certain standards when performing task reviews. Implementing a more encompassing documentation standard early on in a new project, or right away if we were to further develop this application, would help us achieve our own, and our stakeholders, goals regarding technical documentation.

Application of Scrum

Except for the eventual inclusion of a Scrum master, we all had equal roles in the project. This made our work process flexible, but if we were to redo the project we would probably have had more specific roles. Even though all members would contribute to the code, one could be more responsible for the customer value, and one for the quality of the code. Some areas of responsibility came naturally, especially when it came to the APIs or different technical areas. When some members already had done tasks with the Google or Microsoft API it came naturally that the same people kept working with similar tasks, as well as answering questions regarding these areas.

When it came to agile practices, we had one or two standup meetings and two longer meetings each sprint, where one of the longer meetings included a sprint review. We had weekly meetings with our stakeholders except during the last sprint where we did not have enough time for a meeting. In a future project, we will set up a meeting structure and the scrum master role earlier. We noticed that our workflow improved when we had more standup meetings and a designated scrum master.

We did not have an official product owner, but we as a team chose to see our stakeholders as product owners. We spoke to them after each sprint which often led to new user stories or reprioritization of existing stories. It might have been better to let someone in the team be the product owner, but since we did not try we cannot say for sure what would be preferable.

In the beginning we used Glitch.com, an online developer site where it is easy to write code together. Thanks to this, we could all learn the basics of JavaScript, HTML and XML requests. It was an easy way to share knowledge and fast prototyping. We would definitely use the same tool again in an early stage of a web application. Each of our weekly meetings contained a “help section” as a main topic, and this provided the group with a platform where

we could help each other with different parts of the code. More often than not, we would start to discuss how to solve problems directly within the meetings, but in future we would most likely prefer to only discuss them briefly and then schedule a time where we could take a closer look at the issues. Even in the later stages of the project, where we developed locally with Git as version control, we exported our project to Glitch, so that we could code in pairs when we got stuck working on our own.

We always had an open climate regarding asking for help and sharing knowledge with each other. This is probably one of the important factors for the success of our project. Some of us had more knowledge about scrum boards, some of Git and some of Glitch. In our early meetings we discussed how our branch structure should work on GitHub, as well as the structure of the scrum board. Even though we changed and learned things during the course of the project, it was definitely advantageous to create these kinds of structures early.

The opinion of the team is that our application of Scrum gave us useful insight and a clear overview of the project and played a central part in how we produced our application. The Scrum master made sure we followed our social contract, our review system and measured our KPIs. Our short meetings made sure every team member could get help or share insight. Finally, our tools enabled teamwork and the ability to separate features and sprint tasks. If we were to continue the project we would make sure to use all these parts in our application of Scrum. By keeping an open mind, researching about Scrum, having open discussions and reflections we could identify new ways of improving the system, namely by modifying what we already used or by adding completely new processes or applications.

Overall, we were all very pleased with the result of the agile methods we have learned and we look forward to implementing these in future endeavors.