Post-mortem report

Software Engineering Project (DAT255)

Team "Team name"

Ahlstedt, Mattias
Andersson, Patrick
Düsterdieck, Olof
Olsson, Jesper
Strandman, Anton

Table of Contents

1. Used processes and practices

1.1. Agile software development

During the project, an agile software development method like Scrum was used. We have obtained a fair understanding of Scrum from lectures and workshops.

1.1.1. Sprints

Our sprints were decided to be five days long, spanning from Monday morning to Friday noon. In an effort to keep a sustainable pace, the time put in by each member (excluding lectures and such) was expected to be slightly less than 20 hours per sprint for the project itself, while the total time spent on the course was estimated to be 20 hours per week.

1.1.2. Prioritising the user stories

When we prioritised our user stories, our main focus was to identify what functionality was important to the user. We decided to prioritise according to a methodology called MoSCoW, where you categorise the functionality into what the product must, should, could and won't have. We used Trello to manage and prioritise our backlog.

1.1.3. Velocity and estimations

Before each upcoming sprint we estimated the time required to complete each given user story in order to decide which and how many were to be put into the sprint log for that week.

1.1.4. Stand up meetings

At the start of each day, we decided to do stand up meetings. We set the maximum time for such meetings to be 15 minutes. All team members were to tell each other what they had done since the last meeting, what they would do until the next and problems that they needed help with.

1.1.5. Evaluation meeting

Each Friday afternoon, we held an evaluation meeting where we would have a rundown of the product, the work done during the week, discuss the result and, if need be, possible solutions to any arisen problems. The same was to be done for our process.

1.1.6. Product owner

We decided to have a product owner whom we could turn to in order to discuss design decisions, expected behaviors and priorities. The person was supposed to wear a special hat which indicated if he was a product owner or a programmer at the time. Patrick was assigned this role.

1.1.7. Scrum master

A scrum master was selected in order to make sure the team process worked smoothly and any issues which hampered the progress were resolved. Anton was assigned this role.

1.2. Definition of done

At the beginning of the project,we set up a *definition of done.* This was to help us decide when any function/module, as well as the whole project, was in an acceptable state for delivery and should not be further developed. Our underlying reasoning was that a project is never done, since there is always more to be done and we therefore needed to define what should be considered “done”.

Our definition of done consisted of the following points:

- Coding tasks completed.
    - Unnecessary code removed.
    - All code is documented (at least) on the method level.
    - Necessary GUI components created.
    - New functionality integrated with the rest of the application and GUI.
    - Unit testing of model – written and passed.
    - Previous functionality tests reviewed/run and passed.
    - Find bugs/pmd errors corrected.
    - Verified with STAN.
    - (Code reviews conducted.)
- Acceptance criteria (defined in the user story) is verified during testing by customer.
- Defects are in an “acceptable” state to the Product Owner.
- User Manual updated (when applicable).

Our definition of done was abandoned as it was considered unattainable in the face of the short time span in which the project was to be carried out. Instead we focused on implementing functionality. User testing was the primary way of verifying that things worked and when applicable, unit testing was used. The extensive quality requirements were still to be held in the back of our minds but was now considered to be more of a guideline than a rule.

1.4. Testing

The majority of the functionalities were dependant on real life circumstances such as specific WiFi-connections and physical positions. This made unit testing difficult and we decided to mainly focus on user testing.

1.4.1. User testing

We user tested our product by trying out our product in its intended context and making sure everything worked as expected.

1.4.2. Unit testing

In a few cases where we found it suitable we used unit tests. The major reason behind this was that user testing was a harder method to use in that the functionality was one or more steps removed from the user.

2. Time

Below is a very rough estimation of how much time, in hours, each person spent on different parts of the project. It should not be taken as definite values, but as a reference of which parts each person focused on the most.

Activity	**Anton**	**Jesper**	**Mattias**	**Olof**	**Patrick**	**Total**
“Konceptstuga”	6	6	6	6	6	30
Meetings	10	10	10	10	10	50
Documents and product descriptions	30	23	30	21	35	139
APIs	18	2	43	47	12	122
Android	21	4	5	3	35	68
Model	21	56	2	2	4	85
Integration	18	3	12	10	10	53
Bug fix	12	11	18	23	25	91
Workshops + final	13	9	6	13	12	53
Post-Mortem Report	6	18	10	8	10	50
Lectures	22	22	22	22	22	110
Total	177	164	164	165	181	851

As can be seen in the table above, a large part of the time spent on the project was focused on activities that did not directly affect the production of the prototype itself, such as documentation, course activities and lectures. Considering that only about half of the time spent on the course had any direct impact on the development. We believe that we’ve found quite time efficient processes and procedures in order to achieve this.

3. Review of used processes and practices

For each of the used processes and practices we will give an overview of or reflections and answer the following questions:

1. What was the advantage of this process/practice based on our experience in this assignment?
2. What was the disadvantage of this process/practice based on our experience in this assignment?
3. How efficient was the process/practice given the time it took to use?
4. In which situations would you use this process/practice in a future project?
5. In which situations would you not use this process/practice in a future project?
6. If you had the process/practice in a part of the project and not the entire project, how was using it compared to not using it?

3.1. Scrum

After the first sprint we realised that we were not able to properly estimate our velocity nor the time needed for each use case. It was also very hard to match any of the use cases to the short sprint durations. This was mainly due to our inability to properly break down the use cases into smaller tasks, but also since the smaller use cases in and of themselves added no customer value at the end of the sprints.

Upon accepting, in accordance with our product owner, that a use case would not be finished in a single sprint the process became significantly smoother and reduced stress on the team. At the same time we could prioritise building things properly instead of rushing solutions.

What this lead to was that we stopped trying to estimate exactly how long it would take to finish each use case. We still estimated whether we would be able to finish a use case at all using the team's "gut feeling". We also removed the restriction on which use cases were in each weeks sprint log so we could work more continuously towards the end goal. This ultimately resulted in a more fluid workflow with less downtime and bottlenecking, as we could continually add use cases if we were finished ahead of time or all current tasks were already in progress.

Even though our estimations were changed, we still prioritised and reprioritised use cases each week, and before any new use case was selected. This kept the team focused on what was important and what was to come.

An effect of our new estimates was that we decided to choose user stories in a more kanban-like way. We decided to not plan in sprints, but instead pick new use cases from the backlog as soon as another was done. Though, only provided that the new use case did not have a lower priority than any one already in progress.

1. The sprints helped us with defining a definite time for retrospection, of the product as well as the process. Even though we changed our sprints to be less strict on delivery, we still used the end of a sprint to mark time for review of deliveries up until then and see if any changes in the product as well as process was needed.

2. We felt that the delivery of a sprint suffered if the sprints got too short. When that happened the delivery either couldn't be met or we split up an assignment to such a degree that it wasn't a clear benefit to the customer in the end. In some cases these problems might have been circumvented if we would have been better at estimating the needed time to implement use cases and splitting the use cases into manageable chunks (see Breaking down use stories). But even then implementing only small parts of the use case would most likely not have provided any customer value.

3. We believe our process was quite time efficient. Using scrum/kanban helped us plan our todos without any real extra time spent.

4. To divide the project with clearly defined implementation phases is something our team found very useful. What kind of delivery to require after each sprint should probably change depending on the length of the project but in almost all cases it is a natural way to incorporate retrospection into the workflow.

Our new kind of estimates could work in future projects as well, provided that it's used internally in the team (not when communicating with customer as it might seem unorganised), and that the team members feel confident in their own and each other's abilities.

5. An extremely short or small project would probably lose the need of dividing it into sprints since the need for continuous retrospection diminishes.

6. Even though we abandoned the idea of sprints, this was mainly due to the issue of the short sprint duration and not that we found the idea of sprints in any way lacking. Sprints were simply not suitable for a project of this small size and time frame. For this kind of projects we found that a more kanban like was more suitable for us.

3.2. Stand up meetings

We found stand up meetings to be very useful and a good way to keep all team members up-to-date on progress, deadlines and more.

1. The stand up meetings helped coordinate us as a team as they made each member aware of how the product was evolving on a day-to-day basis. The meetings also made it easier to help each other out and formulate problems.

2. The meetings could be infuriating when tasks required plenty of time to solve, as there seemed to be a lack of progress. The meetings did also not reflect the effort you put into your tasks.

3. We found the stand up meetings to be a very time efficient mean of informing all participants of the progress of the project.

4. We would consider using these meetings in future projects where different team members are simultaneously working on different parts of the project. Especially if these parts need to be integrated with each other.

5. We would prefer not to use these meetings when each team member is well aware of what everyone is doing, or if there is no need to know.

6. We used stand up meetings throughout the whole project.

3.3. Evaluation meeting

We found evaluation meetings useful, although sometimes a bit bloated with discussions of detailed solutions.

1. It provided continuous insight in the development of both the product and our team work as it gave the team members time to step back and review what had been done and what was to be done.

2. The meetings could easily take up an abundance of time which could have been spent more efficiently. This was often due to solution based discussions which could be deferred to specialised conversations with relevant parties.

3. The meetings had very high value, but was also time consuming at times, often lasting a whole afternoon for the whole team. Had the meetings been kept shorter they would have been able to provide the same value for a lower cost.

4. We would consider using these meetings again when a new technique is tried out for the first time or when the product changes both quickly and frequently.

5. We might consider not using these meetings in very small projects with very short time frames. Also where team members work closely on a very limited scope, as keeping track of things in such a situation would not require much effort.

6. We used these meetings throughout the whole project.

3.4. MoSCoW

We found the MoSCoW prioritising very valuable and easy to use for this project. Although this kind of planning might not be accurate enough, as prioritising between “Must have”s may require an additional technique.

1. The advantage of this technique was that we all got a common understanding of what the app should be and what it’s not supposed to be, and could define it in user stories relevant to the customer.

2. Creating use cases considered “Won’t have” were not quite relevant and it was hard to know exactly what should be in each category as we did not have a real customer to talk to.

3. This technique was very time efficient, once a scope (time frame) for the given prioritising was set, as we could quickly determine if a use case was relevant at the moment.

4. We would continue to use this technique in planning with a customer in the future were a need of a common ground needs to be established, that is where the developers and the customer need to get the same idea of what is to come and what is to wait.

5. We would most likely not use this technique as the only planning technique in detailed sprint planning. This technique is a bit rough, at least in the way we used it, and may not give enough details into what is to come in the next sprint.

6. We used this technique for prioritising all use cases.

3.5. Breaking down user stories

We found breaking down user stories into tasks a very handy method to get an idea of what has to be done. Although we could have created tasks for user stories long before they were planned to be completed, in order to find common ground for multiple user stories. We also found that we were often somewhat lacking in how well we extracted tasks.

1. The advantage of breaking down users stories comes from the fact that it gives a clearer view of what needs to be done. This in turn leads to the team more easily identifying problem areas, possible work bottlenecks and dependencies between work that has been done and is to be done.

2. The disassembling of user stories isn’t always unique and there isn’t always a straightforward way of knowing when you are finished with breaking down a user story. This leads to the technique not always being as efficient as it could be.

3. Albeit poor usage of the technique from the team, the method of breaking down user stories still served to be a powerful tool with low time investment for the team.

4. We would consider breaking down user stories in all projects in which user stories can be identified.

5. None, unless you count situations where task are already broken down to a suitable degree.

6. The technique was used during the whole project.

3.6. Trello

We used the tool Trello in order to keep track of our backlog. It provided an easy tool to work with which at the same time gave all team members a good bird’s eye view of what has to be done, what is in progress etc. Although a more suitable application for breaking down user stories and keeping track on their parts might be available with a bit more research.

1. Trello was a very simple tool to give the whole team a bird's eye view of what is going on, and what needs to be done, as well as what has been done.

2. The disadvantage of Trello was that breaking down use cases into tasks were a slightly cumbersome experience, and it was hard to keep track on which task was part of which use case.

3. The tool was quite time efficient as it was easy to use (apart from splitting use cases into tasks) and was very handy for keeping track of the backlog.

4. We would consider using this again for the backlog in a similar project, but we might also consider alternative solutions.

5. We would not use Trello to keep track of a backlog in cooperation with a customer. This tool is most useful for the developing team, but could be confusing for a customer to work with.

6. This tool was used for the whole project.

3.7 Other

The partitioning of work tasks has worked well. We tried not to work on tasks too closely related to each other’s task if we could avoid it (there are times when this wasn’t always possible, such as when we began the project and initially set up the GUI). This was in order to avoid bottlenecks.

We always worked together, sharing the same office space. This gave us the ability to cooperate on issues and to keep each other up to date on what everyone is doing and solutions found. Thus trying to keep a higher truck factor.

4. Review of used functionality

Here we will review the non-process specific solutions that we've used and which could be of interest.

4.1. Google maps

For map functionality in the application we chose Google maps. A large part of the reasoning behind this decision was that since Android is owned by Google, they ought to be compatible. Google maps turned out to be a good, powerful tool which eased the development by providing features that we did not need to write ourselves. Examples of this is drawing lines and placing markers.

4.2. Västtrafik API

In the beginning of the project we intended for Västtrafik to be one of the major parties interested in the application, thus we decided to use the official API provided by Västtrafik for acquiring information about getting from one point in the city to another by public transport. While this may have been a valid decision it was later regretted as Google's Directions API provides the same features but with less time invested, and most likely with higher result quality and reliability. This is since the responses from Västtrafik's API is very inconsistent and poorly structured. Västtrafik's API do gives further ability to customise requests, but as we only present the first and easiest route to the user the added customisability is not needed.

4.3. ElectriCity API

In order to fulfill the requirements of the course we needed to use the ElectriCity API. Fortunately for us we had a natural need for the use of the provided information in the API. The next stop information is retrieved from the ElectriCity API in order to give the user information relevant to his/her current position. We also used an API located locally on the router of each bus to find out which one of the busses the user was on.

To be able to test the software correctly and present it without being on one of Electricity's buses a more useful and fully implemented simulation of a bus would have been needed. This is something we should have realised earlier wasn't provided by the API and maybe implemented a solution on our own.

4.4. STAN

In order to keep code structure at an acceptable level we used STAN (Structure Analysis Tool http://www.stan4j.com). The tool worked well and helped us discover and fix technical mistakes .

4.5. Findbugs and PMD

In order to keep good code quality we used the findbugs and PMD tools. Our use of these tools we’re not optimal as we never properly looked into which rules were being used to evaluate our project.

Even though we checked the number of issues continuously throughout the project, and tried to keep the numbers down, we only fixed issues in the end of the project. This is to be considered a technological dept which we ideally should strive to reduce earlier on in future projects, where a definite final deadline is not as clear and such a clean up may never come.

5. Review of teamwork

The team members worked well together, although the composition of the team could have covered a wider set of skills.

The team members quickly accepted each other as equals and became a part of the team, which allowed us to quickly start focusing on the task at hand. There was no problem with asking for help and the cooperation worked well, except for when writing text documents as writing styles clash and there seems to be an upper limit to how many people can work on one document at a time.

The team members were more or less like of mind, which may have hampered our range of perspectives and solutions. We did not know how to improve our collaboration, which resulted in us working independently on tasks and only worked together when helping each other out as problems arose. This resulted in uncertainty in some member whether or not they had learnt what they were supposed to and had contributed their share.

6. Reflection

What we would do differently in a future but similar project.

6.1. Planning

The planning of the project was cumbersome, since we had little to no control over the majority of the project management, due to the project being part of a course. While most projects have to balance and make prioritisation based on cost, time, and result, we had no option of altering the cost nor the time of the project.

One of the main problems was the time. A time frame of eight weeks was very short for the project, since a considerable amount of time had to go to other things than development of the product. Not only does it take time before a group can function as a single entity, but also we had to create project related documents during this time frame. No one had any notable experience of project management before, meaning all estimates of time and staffing were arbitrary. There were several problems with provided APIs, which also affected our disposable time. Since we were ambitious with the functionality of our product, it was tempting to cut corners and accumulate technological debt.

6.2. The Definition of Done

Unfortunately our definition of done was not followed. The definition included a considerable amount of tasks which lowered the frequency of delivery below what the product owner considered an acceptable rate. Given the nature of the ElectriCity Challenge, the project focused mostly on visible functionality which led to code of lesser quality.

6.3. More focus on high-level structure in retrospection reviews

Something that we would do differently is to put more weight in having more high-level structure planning and reviewing. The planning we did in the project was sometimes done before critical variables and potential problems had been discovered, and when these were identified, no new plan was brought forth.

6.4. Increase the truck factor

In this project not much consideration was given to the truck factor except when, on occasion, we discussed solutions to given problems with other team members. This along with the standup meetings have given each and every team member a general understanding of how most of the application functions.

We have considered using pair programming as a way of raising the truck factor and came to the conclusion that pair programming might have been a good solution if the project was longer or if our team’s goals regarding functionality had been lower.

6.5. Breaking down user stories

After our first sprint, we realised our user stories were too large and that we’d failed to break them down sufficiently. Upon realising this we consciously tried to improve our ability of breaking down user stories into smaller tas

7. Workshops

We had the ambition of attending each workshop, even those which did not seem to be relevant for our project, in the hope of finding additional perspective on the domain in which our work was to take place. We chose our representatives based on who were interested in the theme of the workshop. When there was no such person, there was always someone who volunteered. After each workshop, the attendees discussed with the rest of the team what they had done and learned, as well as potential feedback. No immediate issues regarding the workshops arose and therefore no changes are currently necessary.