Final Reflection

DAT255 VT18

@sea (Tugboat)

By:
Emil Gustavsson
Robin Hekmatara
Moa Josephson
August Lennar
Jesper Naarttijärvi
Hanna Norlander
John Sandell
Linus Wallman

Table of Contents

1. Introduction

During the course Software Engineering project (DAT255), our group has adapted the existing PortCDM app for usage by a towage operator. This has been accomplished by removing unnecessary functionality and making the UI more intuitive.

The development process has been based on feedback from the product owners (Sandra & Mathias) and our contact person (Sebastian). We have also used a lot of techniques and strategies that we have learned from lectures and such.

We start this report with reflection about the process of our work. Here we talk about the group dynamics, the agile practices we have used and how we have monitored our process and evaluated it in various ways. Each section in this report ends with some summarized thoughts on what goals we have for future projects and the lessons we've received during this course.

Then we go on to talk more about the application itself. Here we discuss more about the scope of the app, the design that we have used and the tools and strategies that we have used to make sure that the app is of good quality.

To conclude we then have some final thoughts about the project and the course in general.

2. Process

2.1. Group Dynamics

2.1.1. The Social Contract

During the first week, we created a social contract that aimed to guide our work and make the coordination within the group more efficient. It contains rules regarding meeting days, consequences of arriving late to meetings, roles used in the group, Scrum board guidelines and how to use Git/GitHub. Since the first week, we've updated our social contract several times, and mostly the part regarding meeting days due to the troubles of synchronizing our schedules. In general, the social contract hasn't been followed to a sufficient extent. For instance, we've never used the policy about what happens if a team member arrives late to meetings without notice. We have identified some things that might be the reason for this: first of all, we had no real system in place for taking notes on attendance, which led to us not keeping track of who was late and who wasn't to each meeting. The underlying cause to this could've been that we didn't assign a secretary for every meeting. This lack of follow-up has undermined the legitimacy of the social contract and therefore its purpose has not been served.

Another difficulty with the social contract was to agree upon what should be included in the contract. Since none of the group members had used this kind of social contract before, it took some time to understand what it should include. This also means that it was difficult to determine the scope of the contract and the amount of rules that should be included. Maybe it would have been easier to follow the contract if we would have had fewer rules, that were easy to follow up on. These difficulties could be reasons to why the contract wasn't followed in every aspect. In a future project we think the contract would give the most value if it contained few, but well thought through, rules that everyone in the group agreed upon. It will probably be easier next time, since we now have some experience with using a social contract and some problems that may occur.

Furthermore, a lot of changes in the team's work process have not been entered into the social contract but have nevertheless had some important effects. For instance, with less than half the course remaining we decided to focus on working more in large groups. This has led to better communication and has prevented us from getting stuck since there has usually been someone around who has been able to help. We have

also been able to coordinate our work on different features better than before. However, this change has not been included in our social contract.

In conclusion we think that a social contract is a good way to create a common view of how to organize the work in a team, which is why we would like to use one in future projects as well, but we've learned several lessons as to how it should be used. In order to avoid the problems we faced in this course, we think that in future projects a team member should be responsible for the social contract itself and making sure that everyone understands and follows it. More specifically, this means that the responsible person should update the social contract and alert the group when a person violates the rules in it. Clearly assigning responsibility in this manner prevents the problem of people not addressing breaches of the social contract because they want to avoid conflict in the short term.

Goals for future projects:
We want to use a social contract in order to create a common view of how the work should be organized and to make it easy to see if previously agreed-upon rules are not followed, so that we can discuss it in the group and solve any issues. We also want the contract to be easier to enforce and follow than our social contract in this course.

Lessons to bring into future projects:

- A social contract is a good way of creating a common view, especially in bigger groups where the team members don't know each other, but it can be difficult to enforce.
- Make sure that the social contract is followed by making one person responsible for updating and making sure that everyone follows it.
- Fewer and more well-defined rules in the contract make it easier to follow the contract.
- The contract should be continuously evaluated and updated when necessary, in order for it to be well-suited for the current situation in the group and for the current phase of the project.

2.1.2. Roles Within the Team

The defined roles that were used in the group were the Scrum master and three people who were responsible for one KPI each. At first, we decided to rotate the Scrum master role between three different people, but once it got clearer what was expected from a Scrum master, we decided to have a permanent Scrum master for the rest of the

sprints. Since the Scrum master was supposed to communicate with other course groups and continuously update other groups on our work in the “Scrum of Scrum” meetings, it would have been difficult to rotate the role. It would have been confusing for other groups in their communication with us and unnecessary time would have been spent on coordinating between different Scrum masters. Having one Scrum master worked well for us during the project, and our Scrum master participated in most of the “Scrum of Scrum” meetings, which kept us up to date with the other course groups’ progress. With this experience in mind, we would choose a single Scrum master in future projects as well. We will explain Scrum and the role of Scrum master in more detail in section 2.2.1.

In the middle of the project, we removed two of the initial KPI’s. As a result of that, we decided that only one person should be responsible for the KPI that was left. The reason for why we deleted two of the KPI’s is explained in section 2.3.1. In addition to the defined roles there were also a few informal, undefined roles, that came naturally during the project. This includes a person who booked almost all the meeting rooms and temporary roles connected to specific tasks during the sprints.

We realized later on in the course that some of the decisions that were made during meetings never were written down, which sometimes made us discuss the same topic in a later meeting and made it difficult for people who couldn’t attend the meeting to catch up. In order to ensure efficiency in meetings and make sure that no information were lost, we could have defined a role as a permanent secretary, or chosen a secretary for each meeting or sprint. This is a possible way to avoid that these problems occur in future projects. However, due to the fact that we had difficulties during the course to adjust the meeting times in order for them to fit everyone's schedule, it would have been hard for us to have a permanent secretary for all meetings, so we would have used a rotating secretary role if we were to do a similar project.

Goals for future projects:
We want to use roles because there are some jobs that we want to make sure that someone is doing, and clear roles can help to define areas of responsibility. We want to have as flexible roles in the team as possible so that we’re not too dependent on one group member. However, some roles need to be more permanent since they demand more continuity than others.

Lessons to bring into future projects:

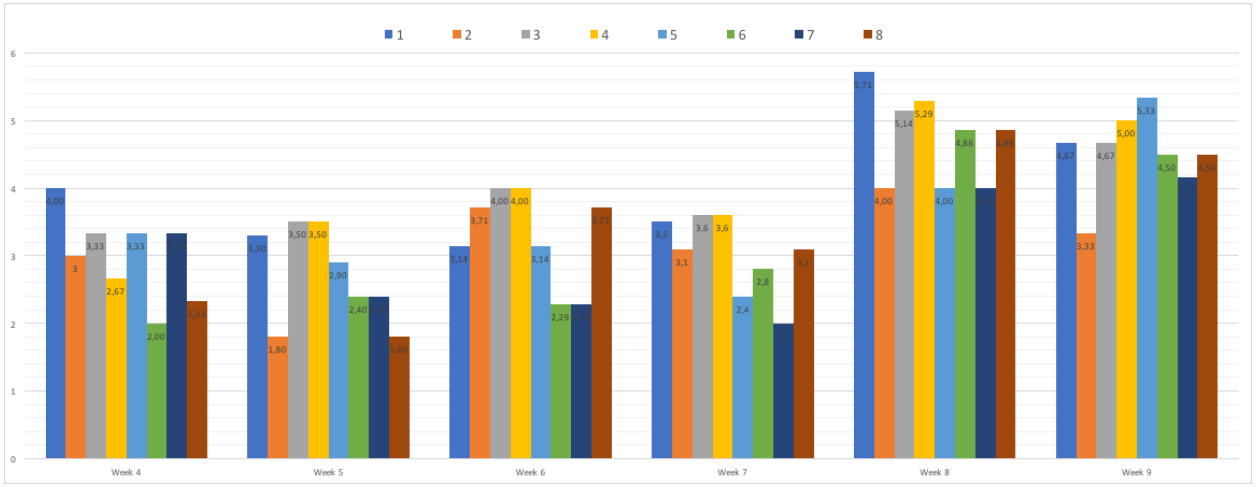
- In order to ensure efficiency in meetings and to create a structured way of taking notes, it would be appropriate to define a secretary for each meeting. This way,

decisions and discussions are documented for later reference, which reduces the time spent on communication and the risk of misunderstandings.

- To rotate between different Scrum masters in the group is not ideal, and therefore it is better to choose a single Scrum master.
- In big project groups like these, it can be advantageous to not have predefined roles since it makes the group more flexible. This can help to address scheduling conflicts and make the group less dependent on one group member (i.e. increases the truck factor).

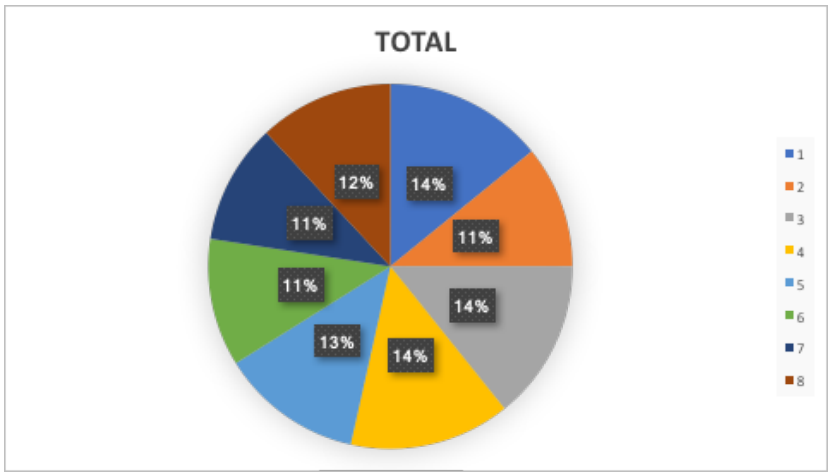
2.1.3. Time Spent on this Course

Table 2-1: Average number of hours spent on this course by group member 1-8 per day during each week.



It has been difficult to estimate the time spent on this course for every week. The team members' hours spent differ each week and all team members have not spent the same total amount of time. The goal was that every team member should spend 20 hours on this course every week, which corresponds to 4 hours per workday. However, no team member reached this goal every week. Instead, the average time spent per person per day was 3,54. However, the average amount of time for the group has increased every week. There is a relatively large spread in how much time ever team member has spent in total on this course, where the highest share of the total time from one team member is 14% and the lowest is 11% (see Table 2-1). There may be several reasons for why the average time spent per person per day has been 3,54 instead of 4.

Table 2-2: Each team member's share of the group's total amount of time spent on this course.



One possible reason for this difference is that the team members may have counted the number of hours differently. We did not define beforehand exactly how the time should be counted. Furthermore, some group members may have overestimated or underestimated the number of hours spent. We have usually approximated the amount of hours spent in the end of the week instead of writing it down daily, which may have caused additional difficulties in analyzing the time. Additionally, there have been three holiday periods during these 9 sprints, which have affected the number of days in the sprints. Some team members have travelled during these holiday periods, which has affected the time spent and the possibility to do pair programming or attend meetings.

Another difficulty, which has affected the time spent during these sprints, has been to build the application in Android Studio. All group members' computers could build the application in the beginning of the project, but after a few weeks, only a few of the computers could build the application successfully. This served as a bottleneck and at times, the group members hindered by technical issues felt that it was very inefficient to spend more time until they could get some help, either with troubleshooting or just being able to do pair programming on someone else's computer.

Also, if a group member had finished his/her assigned task in shorter time than expected, he/she did not always start on a new task. We had a list with "extra tasks" that had low velocity on our Scrum board, that group members should start on if they had time to spare on the sprint. This list was a good idea that we probably could have gotten more use out of in this project if every team member actually used it. However, some group members needed more time than others to finish the same task, which made it difficult to estimate the velocity. The tasks on the "extra task" list usually had a

velocity between 2 and 5 points but were not estimated according to any group member's ability. Therefore, some group members didn't want to start on a new task if he/she was afraid not to finish it in time, especially since these tasks had a low priority.

We are aware that it is not always possible to spend the same amount of time on a sprint, due to differences in the completion time of the assigned tasks, but over several sprints, each member's spent time should average out to about the same. From Table 2-1, it can be seen that the average time per day spent on this course differs up to a maximum of 2 hours between the team members during different sprints. However, all team members' working time follows the same general trend from week to week.

During the last two sprints, there was a significant increase in the number of hours worked. We believe that this is partly due to wanting to finish before the deadline, but we also feel that we had greater clarity about what to do towards the end of the project. This made it easier to work longer hours, since we didn't feel the need to wait for clarification as often as before. This clarity came partly as a consequence of having gained more experience with the existing app and javascript. However, we think the main two reasons for the increased clarity were that we set clearer, more tangible goals for our app, and that we spent more time working together and therefore could discuss any questions right away.

If everyone had spent 20h/week from the beginning, perhaps we would not have had to spend extra time on the last sprint. Although, it is possible that we would have tried to implement some of the other features we wanted instead, and end up having to do more work in the end of the project anyway. In any case, we would probably have gained something from it, however, it is difficult to know what. It is also possible that we simply would not have been able to complete more features even though we had more time since several people could not build the project.

Knowing that every team member should spend 20h/week has probably increased our time spent during some sprints. In future projects with big groups like these, it can be good to have such a "goal" time to strive for and let all members record their time spent. By exposing every team member's work every week, it is more difficult for one team member to sit around and do nothing while everyone else is putting in work.

Goals for future projects:
Our goal for the time spent on projects is that the team members should contribute equally, and that the workload should be evenly distributed between the sprints.

Lessons to bring into future projects:

- It is difficult to decide the number of hours every person should spend beforehand since problems occur during the way.
- It will not always be efficient to spend the same amount of time every week. However, it is good to have a “goal” time to strive for in order to encourage group members to put in enough time. In a future project, we would record the time spent every day somewhere where the group can see it.
- If it would be more strict that every team member should spend 20h/day in future projects, a list with “extra tasks”, easier development tools and a better defined way to count the hours could be useful. It could also be advantageous to have fixed working hours so that everyone would spend the same amount of time but also so the team members are always available to help each other directly. However, it was not possible to coordinate our work this way since all group members had different schedules.

2.2. Agile Practices

During this project, our main focus and overall target was to always be flexible to changes and prioritize the things that created value for the product owner and the end user. In order to reach our aim, we applied an agile approach to the project and more specifically, the agile framework Scrum was used. In this chapter, we will describe how we applied Scrum in this project and how other agile practices have affected our workflow and consequently the end result.

2.2.1. Scrum

As mentioned in section 2.1.2, we assigned the role of Scrum master to one person. We are confident that having a permanent Scrum master is the best solution in projects like these, since it is important that the Scrum master has a good overview of the whole project and maintains continuous contact with other people involved in the project. Our Scrum master has participated in most of the “Scrum of Scrum” meetings, which has kept us up to date with the other course/project groups. Rotating the Scrum master role would require that several group members have a better understanding of the project, which could be advantageous, but it could also lead to information loss and miscommunication among group members as well as between project groups.

Every sprint was one week long, and started with a sprint planning session and ended with a review session and a retrospective. At the planning sessions, we decided what tasks should be done during the next sprint, and at the retrospective we reflected upon

our work process during the last sprint. At first, we decided to have our sprints from Mondays to Fridays, with planning sessions on Mondays and the retrospective and review on Fridays, because these days fit well with our different schedules and the weekend worked as a natural break between the sprints. However, we came to the understanding that the previous fixed meeting days (Monday and Friday) were not ideal, since we had a meeting with the product owner every Wednesday. Therefore, we changed our sprints from Monday-Friday to Thursday-Wednesday, with new fixed meeting times on Mondays, Wednesdays and Thursdays - to make sure we could combine the review with the meeting with the product owner, and write our group reflections afterwards.

An important factor when applying Scrum is to continuously gather the group to make sure that everyone is up to date and identify possible problems, in order to help each other. One of the key methods of Scrum is to gather the team in a short daily Scrum meeting. However, daily Scrum meetings were never one of our goals since the study pace in this course only is 50 percent. We have had difficulties to gather the whole group since we all have different courses and therefore we have had a lot of issues when it comes to applying Scrum, and especially daily Scrum. Since having regular meetings is important we feel that our effort has suffered from this. It has been especially difficult since we have missed some work days due to holidays. One solution that made it possible for everyone to attend at least one of the weekly meetings was to shorten the length of the meetings and meet more often instead. This also improved efficiency, since it was difficult to maintain productivity and e.g. write thoughtful reflections when the meetings were too long. In addition to this, we also tried to be available on the phone if possible, when we couldn’t make it to the meeting. This made it possible for all members to stay updated, which was convenient. Even though there sometimes arose some technical difficulties with having group discussions with people on the phone it was a good solution overall, since it made it possible for everyone to be involved in the meetings more often.

Later on, we also decided that every team member should write a short update in Slack twice a week, to constantly stay updated. Unfortunately, this solution was suggested late in the project (week 7) and therefore we didn’t have time to properly benefit from it.

In projects where it’s difficult to gather the group, we think that the best solution is to have shorter and more frequent meetings, and to write short updates when it is not possible to meet in person. We would also like to explore the possibility of having short remote meetings using for example Skype, since this would enable more direct communication while being more flexible than having to meet physically.

At the presentation we noticed that one group had implemented notifications in their app, which was one feature that we did not have time to implement ourselves. We realize now that we could have asked them at a “Scrum of Scrums” if we could use their implementation if we had known that they had added it. This kind of communication between teams would be beneficial to bring in to future projects, by making sure that the teams clearly communicate what they are implementing in more technical detail than what was done during this project. Perhaps each team’s Scrum master could make a list of added features (and a very quick explanation of their implementation) each sprint and post it in a dedicated Slack channel.

We have realized that applying Scrum is not easy. Keeping a balance with trying to plan things out but also making sure that we try to get things done is not easy in practice. We have likely been planning too much.

Goals for future projects:
We want to use Scrum effectively, in order to continuously create value and be adaptable to changing requirements.
In order to fully be able to evaluate Scrum, we would like to test it full-time instead of part-time like we did now.

Lessons to bring into future projects:

- In the future, we need to make sure to have some resemblance of daily Scrums such as updating on tuesdays and fridays in a text chat so that everyone knows what’s going on and so that people can ask for help.
- Shorter, but more frequent, meetings make it easier for every group member to participate in meetings during the sprint and stay updated.
- It would also be a good idea to have meetings over Skype or other communication channels when everyone isn’t able to meet in person.
- By communicating and sharing technical details with other groups, the development process could be more efficient and groups can share their code in order to create a better application at the end.

2.2.2. Pair Programming

5 out of 8 members in the group come from a computer science background while the rest have a background in industrial engineering and management. Therefore, we have

tried to use the differences in knowledge to our advantage and delegate the tasks according to our strengths.

In the beginning of the project, we decided that we would try pair programming. Initially we stuck with having set pair programming groups for every sprint, but eventually we ran into some issues which led us to rotate between different partners. We thought from the start that it would be practical to try out some agile practices that we had heard about before, which was when we got into pair programming and saw it fit our purpose. Furthermore, we noticed that it was greatly beneficial, and was also used in helping us progress even though we had some difficulties in for example building the application. From the start everyone was able to build the application, but over time more and more members in the group were unable to build the application. However, there were some in the group who were able to build the application throughout the course and we were able to rotate the pairs accordingly and still be able to progress.

In the end of this project, we had to work in even bigger groups since only three group members could successfully build the application in Android Studio. It took a lot of time but still, we learned a lot from each other.

Goals for future projects:
Use pair programming as a tool when appropriate, e.g. to solve more complex problems or for more experienced team members to mentor less experienced members within specific areas.

Lessons to bring into future projects:

- Pair programming can in some cases where the tasks are very straightforward and can be solved individually, lead to a decrease in productivity. However, pair programming can be very beneficial for tackling harder tasks, where there is more uncertainty about how to proceed and where it can be helpful to have someone to discuss solutions with. We think that pair programming was a good choice from our perspective. It helped us quickly get into solving problems and executing tasks and we think that if the future project seems fit, we would definitely try it out again.
- One thing that could be combined with pair programming is a short weekly report of where the group is with the task, if they've had any issues, etc. from every pair. We think that this would make sure that everyone knows where the group is with their task, and if they need additional help with the task.
- Like stated above, we did rotate pairs, but it didn't come naturally to us. Instead, it was used as a solution to a problem that occured and stopped us from progressing. For us, it seemed like an obvious solution to our problem. However,

rotating pairs may not always be the best approach. Sometimes it might even decrease the productivity, as you need to get used to working together with your new partner, that is if you have no previous experience working together in similar scenarios. If everyone is happy with their current partner, it might be best not to switch.

2.2.3. Vertical Tasks

During the first four weeks, we used a horizontal approach when breaking down our user stories into tasks. This was done because we thought that having everyone learn about all the different parts of our app would take too long. The horizontal approach requires that everyone is done with their parts so that they can be added together. Instead of every task creating a little bit of value, horizontal tasks only create value when they are combined.

When using a horizontal approach, we realized that we were very dependent on each other. Therefore, we decided to use vertical slicing of the user stories instead. Vertical slices made us more flexible, since one task could create value even if another task wasn't finished. We also completed a lot more features when we started to make vertical tasks. This may have had something to do with the fact that we dropped some extra features like our own server and database and focused on working with what was possible within the PortCDM system, which made it easier for each group member to understand all parts of our application.

In future projects, we would like to try using vertical tasks from the beginning. This might initially take some extra time since the group members have to learn about the different parts of the project, but the group is likely to benefit from it in the end.

Goals for future projects:
We want to use vertical tasks when appropriate, to ensure that we make steady progress and deliver some value each sprint, and that we are not too dependent on any task or team member.

Lessons to bring into future projects:

- To create good vertical tasks, one should try to make the minimal possible required change in each part of the system in order to get a small, well-defined feature (which is as independent as possible of other features) to work.
- Vertical tasks are better for continuous delivery of value. Consequently, they also make it easier to measure progress than with horizontal tasks.

- Vertical task can serve to give team members a better overview of the project, which makes the team more flexible and less dependent on certain members.
- It is important to evaluate which approach is best for the current team and project, since both the vertical and horizontal approaches have their advantages and disadvantages.

2.2.4. Sprint Reviews

At the end of every sprint, we conducted a sprint review to summarize the outcome of the weeks exercise and meeting with the product owner. During the first two weeks, no sprint reviews were performed. The reason for this was lack of information and understanding. When looking back on the reflections we made after the first two weeks, it is clear that there was a lot of confusion and uncertainty about the Scrum framework in the group. Since we didn't understand how the sprint reviews would work we assumed that the tugboat owner would be the product owner for the project. However, we later realized that this wasn't the case. Although we didn't miss much since we didn't have any sprints the first two weeks.

In week 3 we had our first sprint review. Our initial idea had been that a sprint should run throughout an ordinary week. But during our first sprint review, we understood that we had to change the start and end times of our sprints since the reviews were on wednesdays. We also got to talk to the boat captain in order to get a better understanding on the tugboat situation.

During the following two weeks, we got a lot of positive feedback from the tugboat operator and the product owners on how the app should function. At that point, we decided to change the user interface and create a remote server to keep track of extra information, such as the number of tugboats needed for a port call, because the tugboat operator wanted it.

Later on, we suffered from the two holiday periods that came and because most of us weren't at campus a lot, our productivity decreased. Therefore, we were unable to produce any value to show on the sprint reviews since neither the server nor the User Interface was done.

As a consequence of the decrease in productivity, we decided to scrap the database idea and instead focus on the user interface. We did this in order to have something to show on the next sprint review and so that we would have some solid features on the final demonstration. In the review on week 8 we also understood that we would need a

“send port call feature”, which hadn’t realized before. Because of that there were a lot of features that we had to complete in the last sprint.

It is likely that our productivity suffered somewhat from the fact that we missed the first sprint reviews. If we had focused more on creating small valuable features that are easily presentable we might have been able to actually demonstrate something to the product owners. It’s also likely that we would have learned about the send port call feature sooner. This in turn would have meant that our workload for the last sprint might have been lighter.

Goals for future projects:
We want to have regular sprint reviews so that we’re always in touch with the product owners. We also want quality meetings where we take proper notes and make sure to go through all our current problems in order, which include presenting new features, asking about new features, asking about prioritization and negotiation.

Lessons to bring into future projects:

- We will have sprint reviews regularly even if no new features have been completed. That way we will always know what the product owners want and if any changes should be made. You never know if any in progress features should be scrapped or not.
- We will also make sure to ask about the product owners priorities so that we know which features are most important.

2.2.5. User Stories

We have used a standard pattern for our user stories throughout the whole project. Every story has been labeled with a number and a colour. The related subtasks have then been labeled with the same colour and sub-numbers (i.e x.1, x.2 and so forth). The user stories and subtasks have also been marked with expected velocities. We created a structure for effort estimation in terms of a burn up chart which measures velocity. Our target velocity was 80 points on average each week, which corresponds to 10 points per person. To keep track of the workflow and the user stories we used Trello, which is described in chapter 2.4.1.

We broke down our user stories into smaller tasks, to specify how they would be implemented. At week 6, we came to the understanding that our tasks were not defined well enough, which opened for interpretation of what should be done, and details were missed. Due to this insight, we started making smaller and more specific tasks which

reduced the need for a lengthy description. After this change, we noticed an improvement in our understanding of the tasks, which increased productivity. What we learned from this, is that smaller and more specific tasks facilitate the work flow and minimize the risk of misunderstandings. To be able to create sufficiently small and well-defined tasks, we need to make sure we have enough clarity about what we really want to accomplish.

Another mistake we initially made was that we didn't properly update what user stories were considered done, after finishing all their subtasks. This might be because the user stories were too general and/or comprehensive, and therefore it was hard to see when a user story really was finished. The acceptance criterias we used in our stories should have been even more specific, in order to make it easier to see when a user story was done. Specific criteria would also have made it easier to estimate velocity for each task.

In the start of the project it wasn't entirely clear which features should be prioritized. It wasn't until the end that we knew which features were most important. In future projects, it might be advantageous to prioritize the user stories better in order to finish the most important tasks before spending time on details and extra features. Another important thing when prioritizing user stories is to continuously communicate with the product owner and end user, and listen to their feedback. During our project we have communicated with the product owners on a weekly basis, but we could still have used more of their feedback when deciding which user story should have the highest priority.

The accuracy of our expected velocity for the sprints differed a lot between the weeks. Some weeks our estimations were quite accurate, for example we estimated 80 in velocity for sprint 5 and ended up doing 90 after adding some tasks once we realized that we would hit our velocity target with time to spare. Our effort estimation for week 6 was not as accurate as the previous week. We overestimated what we would get done, despite trying to take into account that we had a shorter work week due to the May 1st holiday. During that week, we ended up completing very few tasks. After week 6 we had this failure in mind, and thought more carefully of how much time each person realistically could spend on the course and how much we could get done during that time. The following weeks were better, and we finished the tasks we were supposed to finish, with a few exceptions. However, factors such as communication and cooperation worked better during the last sprints, which also affected the amount of finished tasks.

Goals for future projects:

We want our user stories and their tasks to match the needs and priorities of the product owner, and to provide the team with clarity about what to work on and how to measure if something is complete, as well as make velocity estimation easier.

Lessons to bring into future projects:

- The most important insight to bring to future projects is the importance of breaking down every user story as much as possible. Having small and well-defined tasks (well-defined by using clear acceptance criteria) makes it easier to understand them, prioritize them and to estimate accurate velocities.
- Finding out what the product owner wants is likely to be an ongoing process; the team should expect there to be misunderstandings about - and changes in - the wishes of the product owner and be prepared to adjust their plans accordingly.

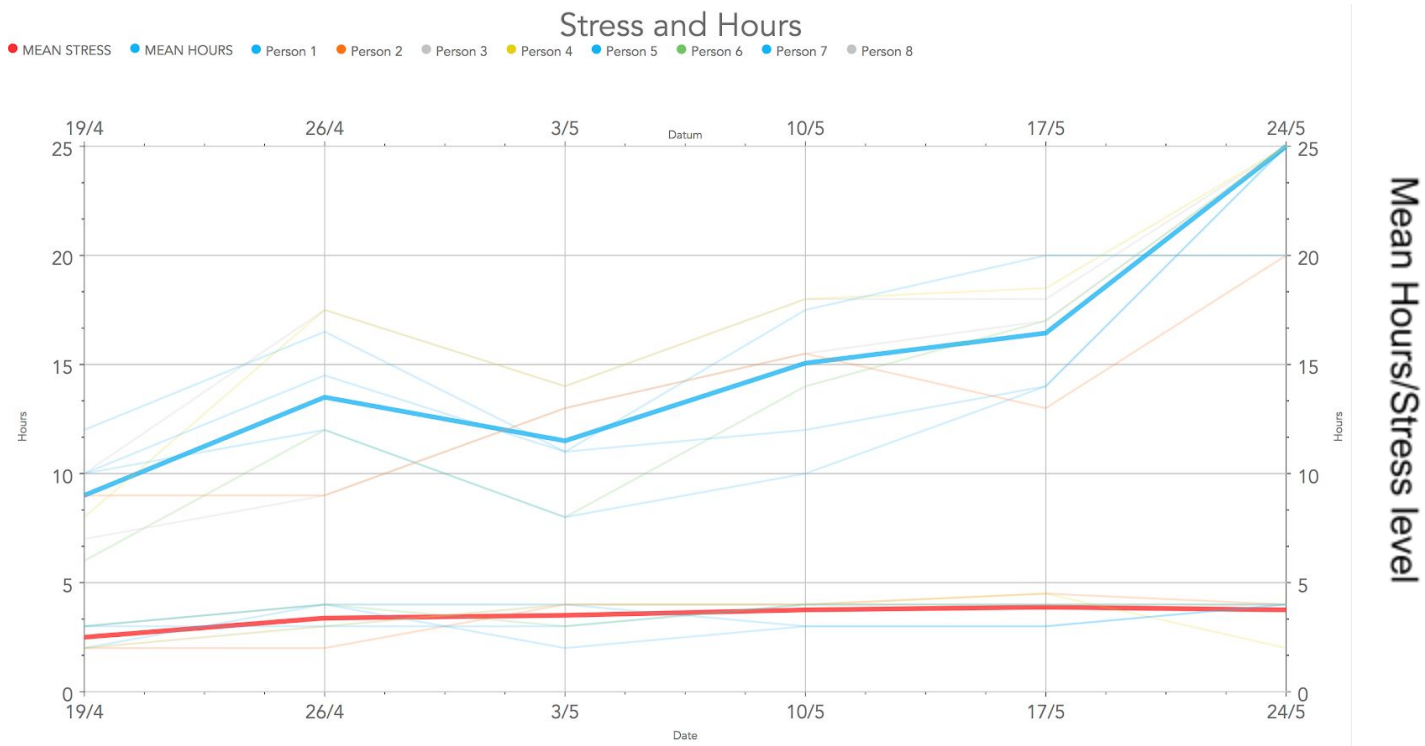
2.3. Monitoring and Evaluation

2.3.1. KPIs

In the beginning of the project, we decided to use the three KPI:s *Burn up chart*, *Code coverage in %* and *Code deletion*. These KPI:s were chosen before we had a proper overview of the project, and along the way we realized that some of the KPI:s weren't the best fit for us. During week 5, we decided to replace the KPI *Code deletion* with a stress index. The KPI code deletion was supposed to measure the amount of code deleted by counting the number of lines committed to Git that were deleted. The purpose of measuring code deletion was to know if we wrote any unnecessary code and try to avoid it. If we would see that a lot of code was getting deleted, we would know that we would have to plan more before we code. However, after our first meeting with the product owners and tugboat operator, we realized that a lot of functionality from the original application/code base would need to be removed. Therefore, measuring code deletion would give a misleading result, which would not bring value to our reflections.

The stress index seemed relevant since these kinds of projects can be quite stressful. The purpose of our stress index was to notice if any group member got stressed out and in that case, help that team member. We chose to measure the amount of stress every week on a scale from 1 - 6, where 1 represented not stressed at all and 6 represented very stressed. We also wanted to investigate if there was a correlation between stress and the amount of time spent on the project each week. The stress level and number of hours spent was reported anonymously in our reflection but visible within the team. From what we can see in the graphs below there does not seem to be a correlation between stress and the amount of hours spent on the project. One thing that we can

note from these graphs is that it seems that the work hours increased a lot as time went on and that the stress levels were fairly low the first week. Besides that, the stress levels seem to have been fairly stable. The stress level for the sprints can be seen in figure 2.4.



Date

Figure 2-4: Stress and hours worked throughout the project

The blue line represents the mean hours each week and the red line represents the mean level of stress each week.

This KPI has been useful even though we did not find a correlation between the two variables. Although measuring stress levels in the team hasn't helped us that much it could still be good to know on a week to week basis since it's important to know if anyone is feeling too stressed out. But the most valuable thing from this KPI was that we became more aware of, and started discussing the hours spent per week (which had been tracked previously, but not really discussed). Since managing time in a project is important this was useful to us. It also showed us that we didn't spend as many hours as we should have. We already measured how many hours we put in each week but it's not that easy to make any conclusions from just numbers. This suggests that it may

have been useful to have time spent visualized from the beginning - perhaps as a KPI, or just as a fixed point on the agenda for the weekly review.

The code coverage KPI was supposed to measure how much of our written code that was covered by tests. The purpose was to minimize the issue of incorrect code, by having at least 70% code coverage. This number was chosen rather arbitrarily in the beginning of the project, and was supposed to be re-evaluated later on. We ended up not writing any tests, and dropping this KPI. Still, we think that Code coverage can be a useful KPI and would probably be more useful if we knew the code base better and had more time. Another reason for why code coverage can be difficult to use is if a lot of features are UI-related, as was the case in our project.

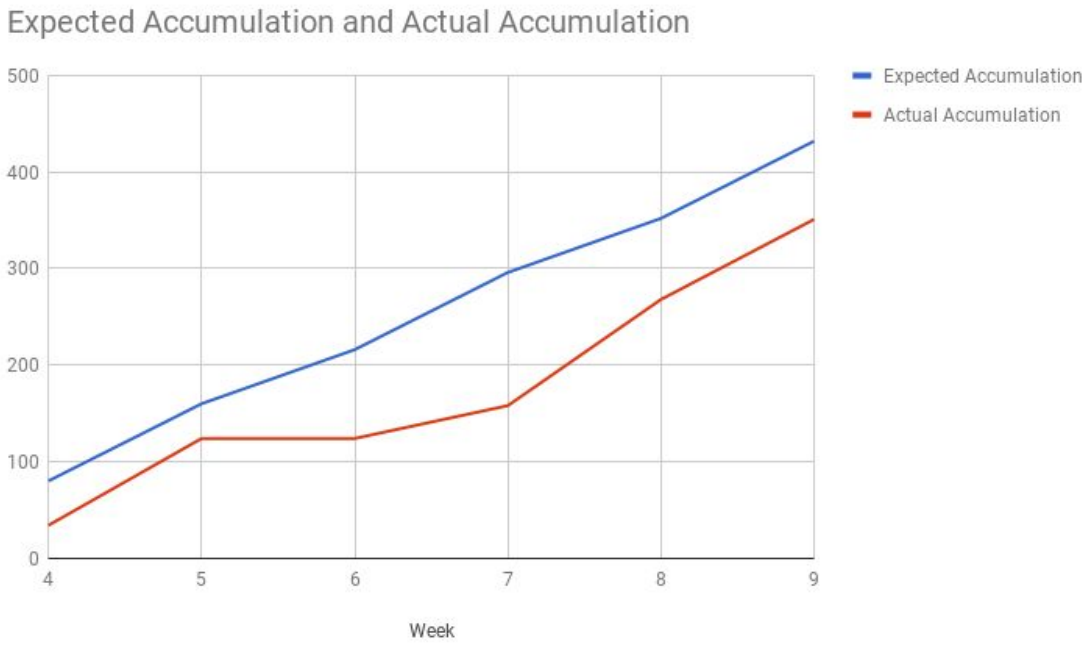


Figure 2.5 Burn-up chart

The only KPI that we have used since the beginning of this project is a burn up chart, that is shown in figure 2.5. The burn up chart measures estimated velocity and actual velocity after each sprint. This KPI was used since we thought that it would be difficult to estimate the task velocities, which might lead us to taking on more tasks than we could handle if we underestimate the velocities of our tasks. We also thought that we would be able to tell if our ability to estimate velocity would change during the sprints. The purpose of the burn up chart was to minimize the gap between estimated velocity and completed points. Our target velocity was set to 80 points per week, which corresponds to an average of 10 points per person each week. The Burn-up chart showed that we

underestimated our tasks a lot during the last few sprints. Therefore we tried to be careful with the velocity estimation by giving our tasks a little higher velocity and making smaller tasks.

Two out of three of our inital KPIs were deleted during the project since they either weren't useful to us or were difficult to implement. It is difficult to know which KPIs that are feasible and valuable beforehand and some KPIs usually get removed during the sprint reflections as we realize that they don't work for us. Therefore in order to minimize the risk of not having any valuable KPI, it is important to start with more than one KPI.

Goals for future projects:
We want to make sure that our KPI's are valuable, well defined and suitable for the project, so that we can get a clear picture of how our project is going, in order to know if there are any problems or if there are any areas that we need to improve.

Lessons to bring into future projects:

- It's important to consider whether a KPI is feasible to calculate or not. Depending on the project, it might not be possible to implement the KPI you're thinking on. Therefore it's important to consider how much time it will take to calculate a KPI and if it's doable at all. It might be difficult to realize which KPIs that are feasible to implement and create value beforehand, and therefore it is advantageous to start with more than one KPI so at least one of them might work.
- Some KPIs will likely be found to not be valuable. Even if code deletion would have been easy to calculate, it wouldn't have been valuable for us since we actually had to delete code to make value to the product owners. Therefore, it's also important to consider if there are cases where a KPI isn't useful. Therefore we will carefully consider which KPI's are useful in the future
- Another way to make sure that we get some good KPI's is simply to make a lot of them. If we do this in the future we can just remove those that don't work and keep the rest. Although this will of course create some extra work in the beginning.

2.3.2. Best Practices for tools and technologies

During the project, we have used several tools in order to structure our workflow, communicate within the group and to save all of our work in a proper way. The tools we have used are listed in Table 2-6:

Table 2-6: A summary of the tools used in the project

Tool	**Purpose**
Git	Version control of the written code
GitHub	Handle our repository and common issues
Trello	Scrum board
Google Drive	Simultaneous work, such as group reflections
Slack	General communication

We decided on some best practices when it comes to using Trello (which we used for our Scrum board), related to how we labeled our user stories and how we used them (see 2.2.5). This worked pretty well when it came to making sure that we didn't get the user stories mixed up. We also added tabs to put the tasks in, such as In Progress, Testing, Review/Verification and Done. This worked fairly well if we consider the technical aspects and it was very straightforward. We also had a tab for extra tasks each sprint where we put tasks that we could do in case we overestimated our tasks and had none left. It turned out that it was unnecessary to have this since we almost always underestimated how big our tasks were instead.

We established a Git workflow that we mostly followed in order to deal with merge issues and similar problems. It can be found in the documents in the GitHub repo. We decided not to use individual forks of the project repo although we didn't update this in the Git workflow. We also thought about having the PortCDM source as a submodule in the project. However, we decided to simply copy the PortCDM project into our repo instead. The reason for this is that it was simply too complicated to implement and would likely take more time than it was worth. Git worked very well for us and we did not really run into any big issues

We divided our slack server into several different channels depending on the topics that were being discussed so that the channels wouldn't be too flooded with text. We had one channel to decide meetings, one to tell when we were available, one general channel and one for technical issues. This worked well when we wanted everyone to know something quickly however it was not useful when it came to information that we wanted to be more permanent. Therefore we wrote some more permanent things in the GitHub wiki instead.

Other than that we also used Google Docs to write most of our documents. It was useful to us since everyone could add to our reflections at the same time. We made some different folders on Google Drive to separate all the different kinds of folders, which worked well.

Goals for future projects
We want to use suitable tools and technologies, such as Git and Trello, in efficient ways and make sure that everyone in the group are comfortable with using the tools

Lessons to bring into future projects:

- In future projects we want to go through our new best practices with the whole group so that everyone understands how to work and so that we know if it's unclear to anyone.
- Another thing to keep in mind is that we might not want to use the same practices that we used here in other projects. Since other projects are different in nature we might want to use our tools in different ways. Perhaps we have a lot of dependencies that get updated a lot. Then we might want to actually use submodules.
- It is also important to try out new best practices. The reason for this is that we won't know if they're good ways to work in or not. There is always room for improvement.

2.3.3. Literature and Guest Lectures

Most of the lectures and guest lecture took place in the beginning of the course, before we had actually started to write code and finish tasks. Due to this, some of the insights that the lectures were supposed to give us, came later on in the project. One example of this, is the knowledge of how to prioritize tasks, that was the main focus of the lego exercise. The importance of correct prioritization became obvious during the last sprint, when we still had one of the tasks with highest priority left. We still managed to finish this task, but in the future it is preferable to finish this task earlier to avoid unnecessary time pressure and/or failure. Another thing that took a lot of time for us to implement was vertical tasks. Because it seemed so difficult to make them we simply used horizontal tasks in the beginning. In the end of the project we understood that vertical

tasks would make the project easier for us, which was also stated in the guest lecture, so we changed our approach in the last sprints.

We definitely learned a lot from the guest lecture on April 3rd though. Specifically we thought more about making sure that our end product brought value (MVP). The lecturer also mentioned that we did not really have all that much time left in the course which also was something that made us reevaluate or approach somewhat. Due to this we started to make our scope smaller. The guest lecture was also quite inspiring and made us feel more motivated towards the project.

Goals for future projects:

We want to learn and use new things efficiently.

In a future project we want lectures, literature or other external information to be considered early in the project, in order to see if it brings value to the project.

Lessons to bring into future projects:

- It might be a good idea to try and consider what has been said in the lectures more in future projects. Instead of simply using the approaches we are used to it might be helpful to try new things that we find in literature and through various lectures. Even though it took us a while to understand everything that was said on lectures we definitely benefited from it in the long run. So in the future we will make sure to reflect on new theories and methods and try using them even if they don't seem that easy to use at first.

3. Application

3.1. Scope (and success criteria)

3.1.1. Scope of the Application

The scope of the application has changed a lot during these nine sprints. The aim has been to adapt the original application to the product owners' and tugboat operator's needs and wishes, by providing the most necessary features and time-stamp information. After the first meeting with the tugboat operator we decided to reduce functionality in the original application, since many of the original features weren't

valuable to him. Therefore, we decided to reduce as much unnecessary functionality as possible and make the application more user-friendly. At that point, we decided to create three main views in the application: one main menu view, one view for all relevant port calls and one view for all incoming requests. Having three main views would contribute to a more user-friendly and lightweight application, that better matches the needs of the tugboat operator.

We also decided to implement a remote server that would help us with custom functions in our application. The purpose of this server was to process some of the timestamp information we got, return a new timestamp after comparing it with our own database, and enabling to store some information that's not accessible in the current application, such as the number of tugboats needed.

To be able to receive a request from an agent with a specified number of tugboats needed for a towage, we needed to implement a way to communicate between the server and application. We decided that several HTTP requests and a database could be used for the communication between the server and database. We also had to request all relevant towage information from portCDM's backend. However, after week 6, we realized that we had overestimated our capacity and needed to reduce the scope of our application. After we set up the server in week 6, we realized that we didn't have enough time to implement the new features that the server would have made possible. In week 7, we also decided that we wouldn't have any communication with the other project groups regarding sending information through the application since we didn't have enough time.

The scope of our final product includes the three views that we developed during the project. During the last sprint we focused on completing previous weeks' tasks and to add features that made it easier to navigate in the application (such as back arrows and a logout button). We also had more features that would have given value to the end user, and in an ideal project all these features would have been implemented. If this project would have been more extensive and if we had more time, we could also have used the functionality that the server and the database would have contributed with.

Goals for future projects:
The main goal with a future project should be to implement the things that the product owner values the most and to make sure that both the working group and the product owner agree on the scope and what is possible to implement before the deadline. If the scope change during the project, the product owner and the working team should both be aware and agree upon the change.

Lessons to bring into future projects:

- To reduce functionality and the scope of the application can also bring value to the end user, which means it is not always better to prioritize new features. It can also be valuable for the end user to ensure a user-friendly application that is easy to navigate in, even though the functionality remains the same.
- It is very important to consider time limits and available resources in order to complete all the tasks with high priority before deadline.
- Continuous communication with the product owner is necessary, so that everyone is aware of the scope of the application. This also means that it is possible to change approach during the project, if the priority changes or the product owner changes their preferences.

3.1.2. Success Criterias

Our first success criteria was that the application should have less information than the original version, and be more lightweight and user-friendly. To this end, we have made a design for a UI in which we reduced the number of views to three main views.

The second success criteria was that we should deliver a product that was complete and functioning; it is better to deliver fewer features with high quality than to have more features which are not that well-implemented. Also we realized late in the project that one thing we needed was the ability to send port calls. At first we didn't have that feature in our app since the end user didn't want it. Although we later had to add it in again since it was necessary for the final demonstration. We decided to skip some features that we discussed (for instance a chat feature requested by the towage operator), and instead focused on making the chosen features to work well. We managed to complete the features that we chose. We made a main menu and made sure that there were only some few views relevant to the tugboat owners.

An important part of the course was to learn to communicate with end users and the product owner, and develop software mainly based on their needs. Therefore, another success criteria was to meet the requirements of the tugboat operator and product owner as much as possible given the scope of the course. We have participated in almost every review session on wednesdays, where we have communicated with the product owner and the other course groups, to make sure that we got the information we needed to develop the application. We have also had two meetings with the tugboat operator. However, we haven't been meeting the tugboat operator as often as we thought we would. The reason for this is that we realized that it would be more valuable

for the product if we put more time on developing the app, since we already had received the input we needed for clarity about what to aim for with our product. After the last meeting we used our time to develop the features he asked for, rather than meeting him again, since we didn't have a lot to demonstrate.

Another success criteria was to ensure that our app is compatible with other project groups' applications that we needed to communicate with, and that our work didn't overlap theirs unnecessarily. Specifically we wanted to make sure that other types of users could add extra info that the portCDM database did not have. However, in week 6, we decided not to use this success criteria since we felt it was taking too much time and we needed to focus on implementing our features. We could probably get value out of these features however there are a lot of technical things that need to be fixed before such features would give the product owners any value. Also it wasn't the product owners highest prioritization so we there were several reasons as to why we removed this acceptance criteria. To achieve this success criteria, while we were still trying to do so, we have had continuous contact with the other project groups. The communication with other project groups has mostly been during the Scrum of Scrums meeting and in the Slack channel of the course. We mostly worked with the agent-teams, since we needed information from them in order to develop the application for the tugboat operator.

Goals for future projects:
We want to have clear success criterias so that we know when a feature is done.

Lessons to bring into future projects:

- Next time we will make more specific success criteria: As an example we will specify how well the app has to work or how it should look etc. We will also continuously reflect on the success criteria and bargain with the product owners if there needs to be any changes to them, just like we did in this project.

3.2. Design

3.2.1. Design of the Application

It took us three weeks before we really had an idea of how to design the application. The main reason for this was that the current application was complex and therefore hard to understand. On top of that, the system with timestamps and statetypes was not as intuitive as we had expected. After four weeks, we came to our first insight that the existing application had a lot of functionality built into it and that we should reuse as

much of that functionality as possible. We realized it was built in a modular way and we got inspired by that and decided to build our own application guided by the same modular principles. The reason for that was in order to make it easy to identify dependencies and to make sure our added functionality wouldn't crash the current methods/dependencies. To make that happen we noticed that we needed to add all additional functionality in our own methods, and try to keep the existing methods unchanged. For example, we followed the current application's structure and put all the http requests into "actions". However, we ended up not following the modular design approach in all of our added functionality since we ran low on time and we had to focus on making the application work.

With regards to what to bring into the future, we've realized that a modular design principle is a good way of solving dependencies but when you work under heavy time pressure the modular design principle falls down in priority. To build a modular application you need time to figure out how to design it, but as we said earlier it's also important not to get too caught up in specifying the design, but rather you at some point just need to start implementing in order to create some value. We've realized that regarding design principles there is always a tradeoff between designing the application in a "perfect" way and creating some real value for the customer in time for the deadline. To solve this tradeoff in the future, we think that it's appropriate to put a limit on how much time we spend on initial design of our project, so that we can get to work faster.

Goals for future projects:
In future projects, we would like to have a technical design which makes it easy to get an overview of the application and makes it easily modifiable and extendable.

Lessons to bring into future projects:

- A modular design is a good way of handling dependencies but it requires a lot of time to figure out how to create and order the modules. If done correctly, with a logical structure and a good separation of concerns, this can make it a lot easier to modify or add to the existing code.
- There is a time tradeoff between designing the application in a "perfect" way and creating some real value. To balance this tradeoff in future projects, we need to make sure not to spend too much time on designing the project architecture in the beginning, but rather decide on an initial implementation which can be changed or further defined as we gain more clarity about the project through working on a few sprints.

3.2.2. Behavioural Overview of the Application

We'll go through this section by first showing our initial idea of the application, then showing the result and finally, we'll reflect upon the differences between the former and the latter.

Since some parts of the original application were user-friendly and our time was limited, we tried to use as much of the existing code base as possible while still taking the product owner and tugboat operator's wishes into account. Reconstructing the whole application would require much more time and would probably not be valuable enough.

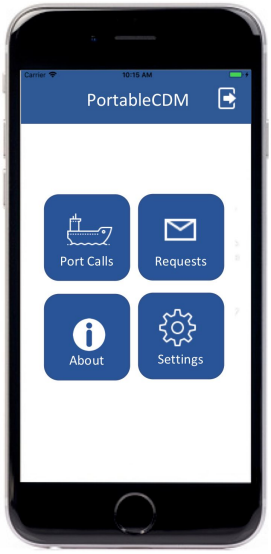


Figure 3.1 - Conceptual model of the main view

The initial idea of the application

In figure 3-1 you see our first idea of the application's start view. We realized, through meetings with our end user, that the current user interface was too complex and not very user-friendly. Therefore, we decided to create a simpler start view from which you can navigate to other views. Our idea was that you should be able to navigate to two primary views from the start view: *Port Calls,* which lists every port call (see figure 3.2) and *Requests,* which lists all requested tows (see figure 3.3).

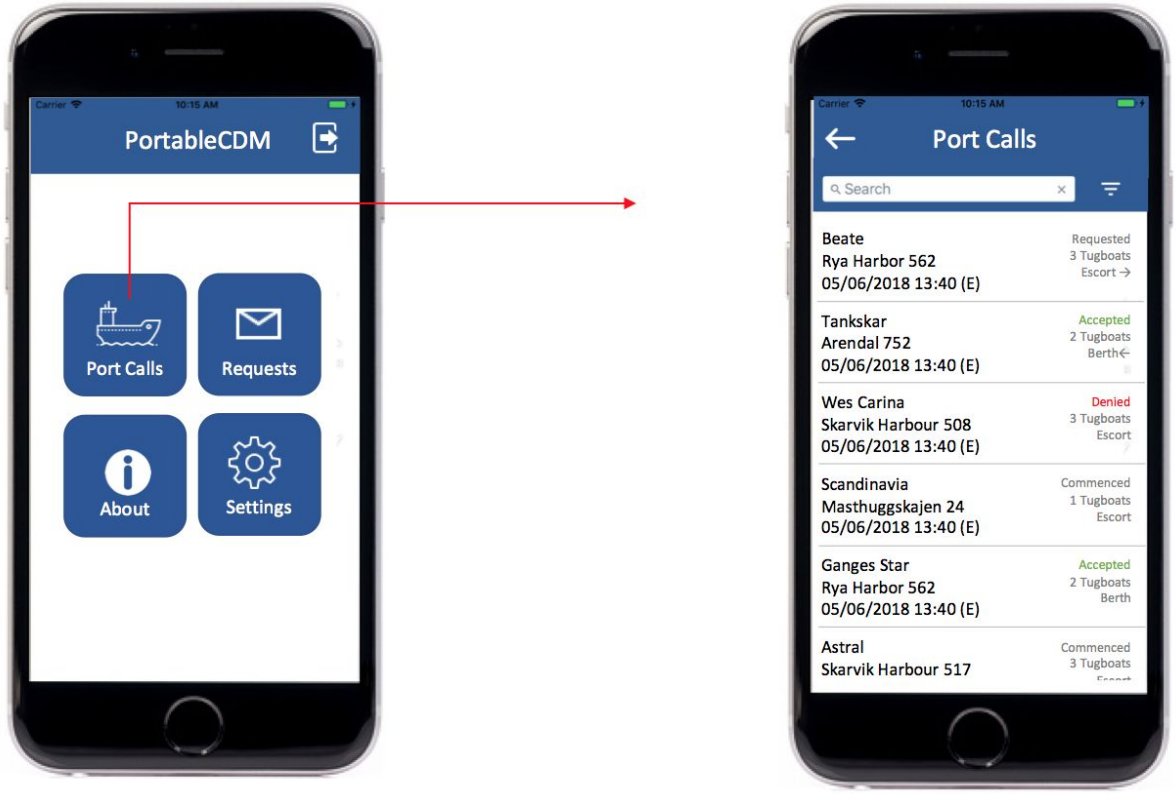


Figure 3.2 - Conceptual model of the navigation from menu to the view Port Calls

In the view *Port Calls*, every incoming port call to a specific harbor should be listed together with information about ETA, berth, towage status, number of tugboats and type of job (escort or harbor/assistance towage). In the Requests-view, every port call that has a state type that includes *Towage Requested* should be listed together with the same information as in the view *Port Calls.* In addition to providing information about a requested towage, our idea was that the user should be able to accept a towage request from the Requests view.

When clicking on a specific port call in the *Requests* view, the user should be able to get more information and further change the status of the port call. The idea of the flow through the application is visualized in figure 3.3 below.

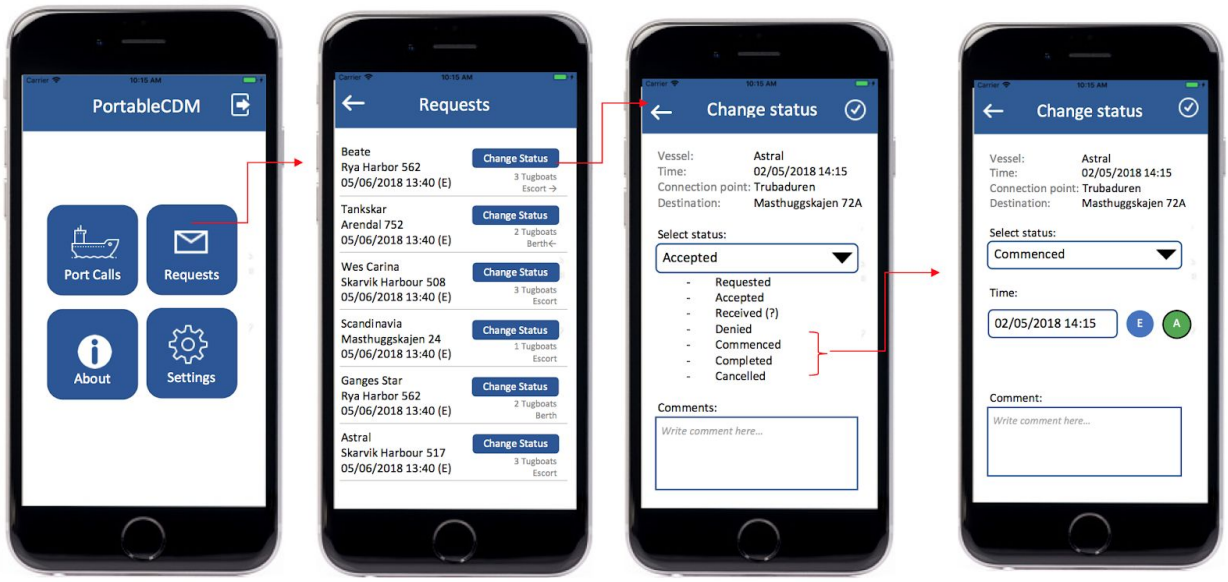


Figure 3.3 - Conceptual model of the navigation from the start view, through the Request view and further on.

The resulting application

In figure 3.4, the resulting start view is visualised. It looks pretty much as we expected besides from the size of the button icons and button titles. One change that we did from our original idea was to have a view that not only includes requested tugboats, but also all other state types that includes tugboats. The extended *Request* view is now called *Tows* and is accessible from the start view. In the Tows view, all information regarding berth, ETA, status on towage, number of tugboats and type of job (Escort or Towage) is included. Regarding the information about number of tugboats, since we didn't implement our own server/database, it was not possible to add this piece of information to the application. We included static data anyway to show the product owner our idea. When clicking on a specific port call in the towage view, the user is provided with additional information, for example: connection point for towage, start time for towage and whether the most recent timestamp is actual or estimated.

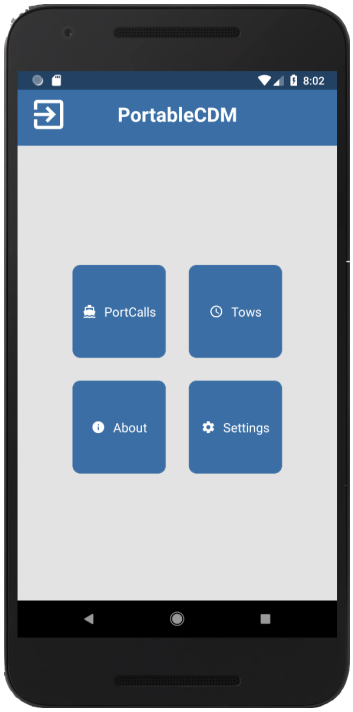


Figure 3.4 - Screenshot of the main view

From the view with additional information, the user could choose to either accept a request directly or report another state type. Regarding the Accept-button, we didn't have time to implement this function, but we implemented the button anyway since our end user requested it. If the user chooses to report another time stamp, the original favorite state-view will be shown, which by default includes all state types regarding towage. When a state type is chosen, the user will be navigated to a the original view where timestamps are reported. The whole flow throughout the application, as described above, is visualized in figure 3.5 below.

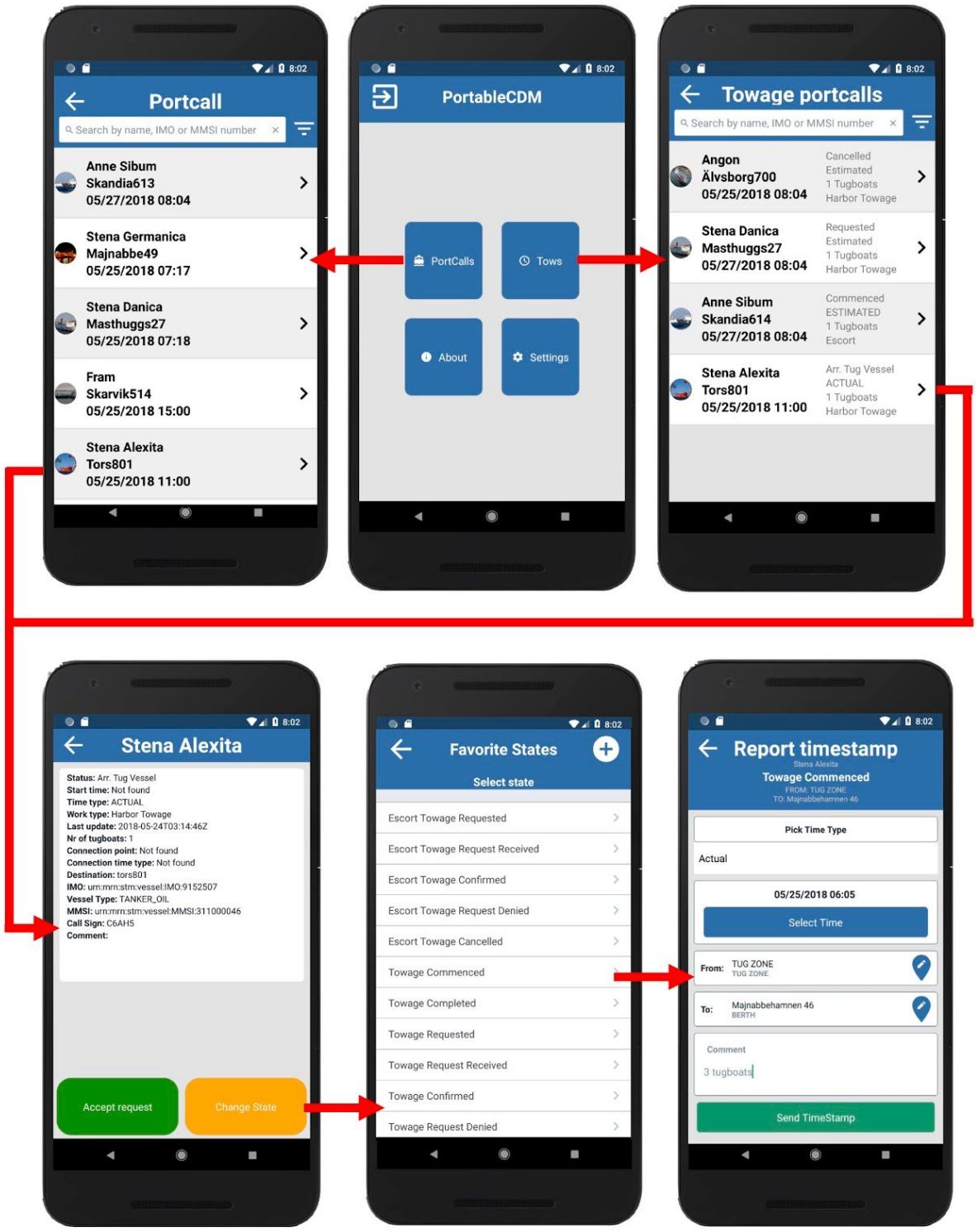


Figure 3.5 - Screenshots of the resulting application with arrows showing the flow through the application's views

Since most parts of our idea was implemented, we've come to the conclusion that creating a conceptual idea of the user interface is a good way of steering the work processes. There are two main reasons for this. First, it's easy to visualize both functionality and user friendliness to the product owner, which forms a good basis for feedback. Second, it guides the work process in the group and makes it easy to split up tasks among team members. We see a big need of continuing to create conceptual ideas of user interfaces in future projects. With that said, we also realized that some parts of the views that we wanted to create were not possible because of the lack of time and the complexity of the time stamp system. Therefore, one shortage of this approach is that it's hard to get a good understanding of the underlying technical complexity. This, in turn, complicates the estimation of working hours. To tackle this problem we've come to the conclusion that it's crucial to constantly reflect upon what technical implications a specific view generates.

We've learned that it is important to focus on the things that actually create value. If the tugboat operator values a user-friendly application, it is not relevant to spend many hours on creating extra features that aren't demanded or minimal design improvements (like trying to make the icon and text on the buttons a little big bigger as we first wanted), since it does not create enough value.

Goals for future projects:
We want everyone in the group to have a common view of the end product and work towards it. A conceptual model of the user interface is a great tool to visualize that goal and minimize possible misunderstandings.

Lessons to bring into future projects:

- Creating a conceptual idea of a user interface is a great tool that makes it easier to collect feedback from the product owner and guides the work process within the team in a good way.
- It can be hard to understand the underlying technical complexity from a conceptual idea. To handle this difficulty, it's crucial to constantly reflect on the technical implications a specific feature generates.
- The conceptual model will most likely change along the way and it is important to expect this and be flexible. It is also important to consider if small changes create enough value to implement.

3.2.3. Structural Overview of the Application

For the structural design of our implementation in the application we wanted to use many of the already existing design patterns. For example, splitting internal and external requests into different modules, and importing external requests when they're needed, to make it clear when we use external requests. However, we did not keep the original applications structure to the fullest. This is because we did not understand how everything worked at the start of the course, and needed to start implementing code so we could complete our sprints.

To get a better understanding of the codebase, we decided to create a task during sprint 4. The task was to implement a UML diagram over the project. After creating the UML diagram we realized that it did not help us understand the codebase better, but we got some more insight in what the main dependencies are and how we will structure our implementations. The reason the UML diagram did not help us understand the project is mostly due to how Javascript modules work. Once you import a method, it's available within that module and all other modules that extend this module.

When planning the 6th sprint, we discussed if we should created a new diagram, more specifically a sequence-/use case diagram over how the user interface and code worked together. We decided that we would not created such a diagram because we only had three sprints left, and had plenty to accomplish before the final review. If we would have had more time, or brought this up earlier in the course we would have created a flow diagram to show how the user interface and code worked together, and what kind of data that's shared between them.

A contributing factor for us not having a concrete structural overview of our application is time. In the start of the course we had trouble understanding the codebase and its structure. All of us had to focus on learning react, and most of us also had to learn javascript. So when we started implementing our code we were already in week 3, and still had trouble understanding the existing code. In a perfect project we would all be familiar with the coding language before the project starts, and then have an easier time implementing a structural model.

Goals for future projects:

We want to have a structural overview that gives us a high-level understanding of how the application/system works.

Lessons to bring into future projects:

- Take the time to implement a structural model to follow. Be ready to change it if appropriate.
- The ideal is to have a good knowledge of the code language before the project starts. Otherwise, try to take the time you need in the beginning of the project to get a better understanding of the language. It's hard to understand what kind of diagrams/models that will be useful if you don't understand the language.

3.3. Code Quality

3.3.1. Acceptance Tests

As we did not implement anything that could be tested as/by a user until very late in the project, most of the acceptance test has been performed by the team. We did check with the tugboat owner if he liked it. We also demonstrated what progress we had made in each sprint at the Scrum review session. This is the extent of acceptance test we performed with people outside of the development team. Setting up tests with possible users each sprint would take a lot of time that could be spent on developing the app which is one of reasons that we did not do such tests.

As for the testing that was done within the team, we made sure that the views displayed correct information, that the application navigated correctly between views etc.

Doing acceptance tests with someone outside of the development team would certainly help to find problems in the UI that we as developers of the UI would have difficulties spotting. This would be a good next step if we or someone else were to continue working on the application.

In order to be able to perform acceptance tests there need to be time available at the end of each sprint. So it is necessary to take into account the amount of time that is needed to perform the acceptance tests for the user stories that are selected for that sprint. Another necessity for performing acceptance tests is to work with vertical user

stories as that ensures that there will be something to test, given that the developers finish their the user story in time.

Goals for future projects:
Perform acceptance test whenever a new feature has been developed, before accepting it as completed.

Lessons to bring into future projects:

- Take the time it takes to perform acceptance test into account when planning a development cycle.
- Work with vertical user stories to make sure that the application can be tested continuously.

3.3.2. Code Quality Tools

We did not use a tool for finding bugs and testing the code at any point throughout the project. This was not a conscious decision but rather a consequence of us struggling with producing value for the product owners and thus we did not have time for doing more extensive testing than just making sure that the application would run. We did develop the app in a modular way which should decrease the number of bugs as the code is more decoupled and independent. We also tried to write in the same style as the original project to make sure that the whole project is easy to navigate through.

One of the reasons that we struggled with producing value was that we worked on making a database so that non-tugboat users could send info that wasn't already in the PortCDM database. Since we scrapped this idea we never got any value out of it.

As we ended up with bugs that could crash the application by the end of the course, it would seem that more extensive testing of code quality would be useful in future projects. However we were not exactly sure how to test the app since the majority of our features had to do with the view. This meant that we couldn't really use unit testing but we would likely have to test the app manually instead. As mentioned earlier we were running out of time in the end and since only one person could build the project during the last few days we did not have time for such extensive testing.

Goals for future projects:

We want to know if our project has bugs early on so that we don't get a lot of accumulated technical debt, and don't have to spend unnecessary time fixing bugs when we assemble the different modules into our final product.

Lessons to bring into future projects:

- Ideally, we would like to spend more time on learning the new programming languages and the best tools for a language before we start working on the project.
- One reason that we never made any tests is because we started too late when it came to thinking about testing. In future projects we will try to implement testing earlier so that we have established ways to test our code once we get to later stages of the project, when we have to hurry and get new features done.
- Another way to get more tests and better code quality in the future is to take on smaller tasks since that means that we will have more time for testing every week. When we estimated the velocities of our tasks we never really considered how much time testing a feature would take.
- In bigger projects code quality is also extra important, since the bigger the project, the more bugs there will be. Therefore we will also make sure that we will find fitting code quality tools and testing tools for our project earlier in our future projects.

4. Conclusions

This course has been a good learning experience for us. For many of us, it was the first contact with Agile methods and Scrum. It has not always been easy to apply our new knowledge but we have become a lot better at applying Scrum.

In previous projects, we have often spent a lot of time on planning in advance how our programs should be structured. Flexibility hasn't really been all that important. Rather, the focus has been more on good design and using quick algorithms. Therefore, it has been valuable to us to learn a way of working where continuous delivery of value is the focus.

In order to coordinate our work in such a large group, we spent a lot of time trying to stay on the same page with regards to our goals and the progress of our work. If we had taken the course full-time, we would have been able to have daily stand-ups and have longer meetings with the entire group whenever necessary, instead of having to deal with conflicting schedules. A way to reduce the communication overhead might have

been to prepare our meetings better by having a clear agenda, and to take more notes during meetings. It would have been appropriate to spend a bit more time on discussing the issue of communication at the start of the course - being such a large group with different schedules, we should have anticipated that we might have some problems with communication.

We had to lower our ambitions quite a lot as the course progressed. In the beginning of the project, we aimed to have our own server and database to add some extra functionality relevant to the end user. We also aimed to have some form of cooperation with the agent group, where our apps would be able to interact. As time went on, we realized that we would not be able to reach these goals, and instead had to reduce our scope quite a lot. Maybe we should have started at the other end, and implemented more UI-related features such as only showing towage port calls first. Then we could have left the possibility of adding more advanced features open until later in case time permits it.

Starting with such high ambitions may have contributed to the problem of too much time spent on planning. In the first weeks we ended up spending too much time on reading and trying to specify the things that needed to be done rather than following the Scrum philosophy of continuous iteration and adjusting course from sprint to sprint. Later in the project, we realized that we had to plan less and just get started on implementing some important features instead. Rather than planning big features, we started to focus more on completing a minimal, realistic set of features to make sure we would get something done. This worked really well and we soon started to see tangible results. We were able to finish our tasks (although they were not that polished) in time for the demonstration. Even though it took some time for us to properly start applying the agile methods, we have definitely learned from this and feel that we are ready to put them to use in the future.