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

<u>What has been:</u>

In this project, the two main actors for whom we were creating value were the product owners and the client/end user (the terminal). From the start, the main actors we were creating value for were quite clear. Therefore, we had a good picture of which people to talk with in terms of what was expected from our take on the application. However, we naturally also indirectly created value for other actors in the port - by contributing to the terminal's ease of use in terms of the application.

To begin with, the application was briefly introduced by the product owners during the introduction to the project, together with the scenario and us being assigned to a terminal. However, due to the fact that we did not know that much about how a terminal operates from the scenario, it was still a bit unclear in the beginning in what was expected from us - both in terms of features and how to code these features into the application. Therefore, we had to gain some knowledge and come up with ideas of how we would expect the application to be in context with the terminal's needs and we tried to figure out user stories ourselves.

However, after our first meeting with Inter Terminals during week 5, we got a much better grip of what they would like to see in the application and most importantly, how a terminal operates and what kind of role they play in the port. After this, we got a better picture of how the priority of features would be from the perspective of the terminal and we could also connect these with the previously made user stories. Together with the product owners, we developed the user stories further and also prioritized these with respect to what both the terminal and also the product owners wanted in the application.

From here, we developed three epics and those were divided into user stories. The three epics were: showing NOR/ETA in the port call list, a berth view showing current vessels for that berth, and a calculator for calculating ETC depending on a specific cargo type and volume. The epics included more functionalities than we managed to produce, we only managed to finish the basic versions of each epic. The prioritization of these epics were made in conjunction with the end user and product owner.

Due to the lack of time in the project, not all user stories were finished and therefore our scope changed as the project progressed. We continuously calculated if our scope was reasonable by taking all of our estimates and comparing to our velocity. This metric was more or less accurate depending on our estimation accuracy and velocity consistency. When we did decide to change the scope we made sure to bring it up during our weekly reviews with the product owners. The change of scope happened because the estimates were hard to make (due to technical difficulties etc.) and the time consumption was higher than estimated. Therefore, the scope was narrowed down and the user stories that seemed redundant were removed in conjunction with the product owners.

<u>What to do better in a future project:</u>

For future projects, what we have learned from here is that even if you do not know much about an upcoming project in the beginning - you can still start to process it and try to familiarize with the scope of the project and think about what kind of value you want to bring forth to the product owners and the client, in order to clarify the scope in some ways early. The important part is to break down your thoughts and assumptions into user stories and always keep communicating with the product owners so that you are in line with their scope as well and have constant reviews made by the product owners to keep you on the right track. By looking back to the previous reflections from the different sprints, especially the first sprints in the beginning, we can see that we struggled quite a lot in making up our minds on how the scope of this application should be. Additionally, we feel that it is important to always check that the scope of the project is feasible and that we don't have a too broad or too narrow of a scope. Otherwise, the product owners might get the wrong impression of what is actually possible to do in the limited timespan of the project.

How to bridge the gap:
To bridge the gap between our lessons learned and what to do in upcoming projects, we know that even though you have not established your communication with the client - you can still start to digest the project and start thinking of how you can provide with value together with the product owner. That is essential in moving the project forward and not wait for the client, in order to start producing value and to get grips of the actual problem. Later on, you can get useful insights from the end user regarding your existing user stories and scope - which will then be changed together with the consent of product owners. Thus, the development of the user stories and the scope will be an iterative process together with the stakeholders. Furthermore, the use of our KPIs is also important to keep track of the scope and if it is reasonable - thus by using estimation accuracy and velocity consistency (in our case), the scope can be analyzed and adjusted if needed.

****Your social contract****

What we did:
In the beginning of the project the group agreed on a social contract, which can be fully read on our Github repo. Thus the contract was initially based on the expectations on the project. However, during the course we have if needed revised the contract. Examples on things written in the contract is that we are supposed to be in time for meetings and that we all put down 20 hours each week. Furthermore the group has agreed on things not put in text as these things can be expected without writing it in ink. An example of this is how formal the meetings are supposed to be.This is primarily based on mutual understanding of everyone's intentions with the project. As we in the group already knew each other it was easy to get a good working environment in the group. The social contract was therefore not a central part in our work.

What to do better in a future project:
For future projects we think that it is important to agree on a social contract early in the project. If the members know each other well this may work as in this project where much of the agreement is unspoken. If the group does not know each other well before, they can benefit from writing a thorough contract in the beginning, stating some basic foundations for the group dynamics. This will be easier if the group is gathered early in the project. Of course it can be hard if the work process is unclear and thus difficult to estimate the

expected effort of the group. Also it is important to hold people accountable to what is stated in the contract in both good and bad situations. An example of this is having rewards written into the contract for when people do remarkable things or serious talks when the contract is broken multiple times. That way it is not only a negative thing. All in all, this will improve the group dynamic. The unwritten contract should also be agreed upon so that the entire group understand each other.

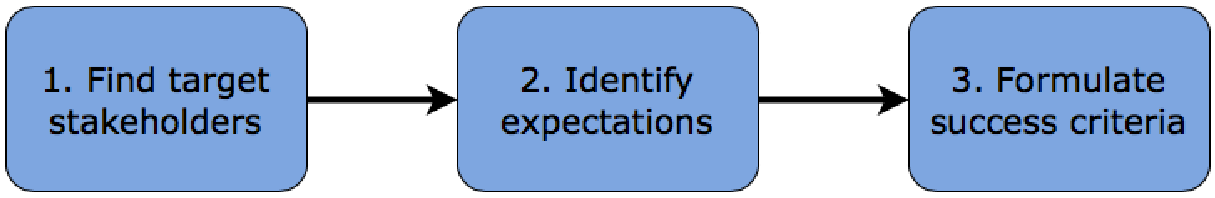
How to bridge the gap:
In order to achieve a good working environment for the group it is important to not only write a social contract but also make it a natural part of the collaboration. It is therefore vital that everyone sees the potential and benefits they can gain from such a contract. A future group will possibly not know each other well in the beginning and could benefit from getting to know each other at an early stage of the project. It is therefore suggested that the team members participate in group activities that could help the group break the ice and get started. One example of such an activity is frisbee golf, which has been the foundation for our teamwork.

****The success criteria for the team in terms of what you want to achieve with your application****

What has been:
At first, our main success criteria was to satisfy the project's stakeholders. Among many stakeholders (including ourselves), we focused on satisfying the final user, and the product owners. Luckily, their expectations were almost identical. This success criteria was established early through internal (within our team) as well as external dialogue with each respective stakeholder. After deciding which stakeholders to prioritize, we sat down with them (the product owners and the terminal worker) to establish what expectations they had for the final product. These expectations were later transformed into a more concrete success criteria, specifically in terms of user stories that had to be completed. We then aimed to complete these user stories as a means to achieve our main success criterias.

In our case, we were lucky that the product owners and the final user had aligned expectations. If this wasn't the case, then we would have to further prioritize which expectation to meet, and one of the stakeholders might be dissatisfied.

During the middle stages of the project we specified our success criteria as completing all of our user stories. We quickly realized that we overestimated our velocity, so we readjusted our success criteria accordingly. After a few sprints we redefined our success criteria as completing four user stories (one per coding pair). We also made sure that the product owners were OK with this change.



In the end we completed three user stories which doesn't meet our success criteria. Nevertheless we think that our way is an effective way of establishing success criteria. We had one underlying, but more vague success criteria (satisfying the stakeholders) and one more concrete, but still related one (completing four user stories).

What to do better in a future project:
For our next project we would go about in a similar way, namely by identifying which stakeholders (probably the product owner) to focus on, deriving their expectations and forming success criterias from these. We should also aim to have a product owner who is better integrated into the team and has a clear product vision, in order to get reviews done by the product owner in a easier way. This would help us specify a more accurate success criteria, and to keep it up to date throughout the project.

One of the challenges we faced during the project was that the product owners didn't formulate a clear product vision, instead they gave us some suggestions of which of our user stories to prioritize. For our next project the product owner role should be better integrated into our team and he/she should have a clear vision of what the finished product should look like. In this case, we would simply specify the overarching success criteria as "Satisfying the product owner's vision".

Finally, changing the success criteria halfway through the project undermines the concept of long term success criterias. It would certainly be preferable if we specified a doable success criteria from the start, but in projects like these, predicting the future is almost impossible.

How to bridge the gap:
We would like to bridge the gap by having extensive communication with the relevant stakeholders from day one. Identify the product owner. Derive expectations from the prioritized stakeholders, and transform it into a quantifiable success criteria. Follow up on this success criteria, and adjust it if circumstances change (make sure that stakeholders are OK with this).

****Your acceptance tests, such as how they were performed and with whom****
What has been:
The concept of acceptance tests only became understood by the group in week 3, at which point we created a plan to formulate acceptance tests for our user stories. Our user stories were however still very unpolished, and they were improved and made clearer before work on acceptance tests began. The real formulation of acceptance tests took place in week 5, when we defined two sets of acceptance tests. These were to be tested by another code pair (which didn't code the part to be tested) in the group. The first test was a general checklist for the definition of done for all user stories, which involved the following criteria:

- The code is clean

- The code has comments explaining the code
- The product owner approves

This test made sure that we could set a user story as completed after all the above points are fulfilled.

The second acceptance test involved individual checklists for every user stories, which served as instructions for the product testers, to make sure each vital function of the app works properly. The points on the checklist were formulated based on the end user testimony of how they would use the functions of the app based on their user story.

Acceptance tests were the topic of the next sprint review, where we further added a point onto the definition of done test:

- It can handle all variations/combinations of input data that it should.

The group realized that this point had to be included, since the tester could mistakenly approve a faulty implementation of a user story. In addition to this, this point would need to be broken down further into definitions of input and acceptable output for each user story.

However, our group soon realised that completing user stories could not be completed during one week sprint process, and acceptance testing was put on hiatus until we could figure out the product code and actually create something that could be tested. This took so long, that the first user stories were only ready for acceptance testing during the last sprint. Our group then realized that the product owner would not be able to check the final version of our work in time, so we could only rely on our own understanding of the app, and hope that the product owner would approve during the product presentation that was held in the last meeting.

<u>What to do better in a future project:</u>
The main learning outcome in regards to acceptance testing is to weave it into even smaller portions of the project, not just user stories. The reason our group couldn’t reflect much on our acceptance testing, was because our user stories were difficult to code, and took so much time that acceptance testing was basically the last thing we did before we turned in our product.

Therefore, if acceptance testing would exist earlier in the process, we would be able to reflect more on how to benefit from them and find potential errors in the code much sooner. We also learned that the broken down tasks for each user story could serve as testing criteria for the product testers, which makes testing more clear and simple to follow. These two things are something to take away for future projects.

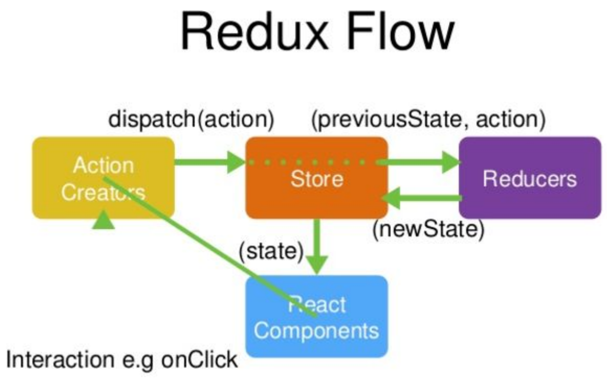
<u>How to bridge the gap:</u>
Acceptance testing can be done earlier in the project by setting aside the last sprint week for acceptance testing, which means that our code has to be finished earlier. Also, by creating testing criteria similar to the broken down tasks of each user story, we could accomplish

more structured acceptance testing. Of course, the ideal situation would be to finish the user stories faster, in order to allow there to be more time for acceptance testing.

****The design of your application (choice of APIs, architecture patterns etc)****

What has been:

The design choices for the application were quite limited since we had to work with an already existing app and codebase. No other APIs were used except for the one already existing and used in the app. One possibility was to create an API for other actors to be able to interact with the Terminals private database, but the feature wasn't requested by any of the actors, neither the terminal nor other groups. In the last week we found out that the app was using a redux architectural pattern, which consists of actions taken in components which are then dispatching information through reducers and finally being retrievable from the state.



What to do better in a future project:

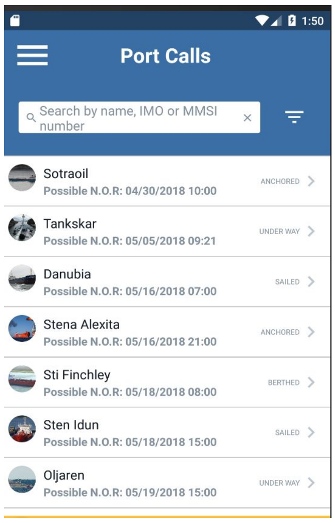
One thing that would be useful in future projects would definitely be to get familiar with the architecture earlier. Since we are used to being given directions in class we assumed that there would be an introduction to the software or some tutorials, and therefore also an opportunity to learn about the architectural patterns, but there wasn't, and unfortunately, we didn't take any initiative ourselves to fix that. An earlier introduction and taking the time to learn the pattern earlier would save us much time down the road (we would have a much larger relative velocity a bit into the project, sacrificing some initial velocity). Instead, we had to experiment and try things in order to find out how the app works, and only got proper information on Wednesdays during meetings with Pontus.

How to bridge the gap:

The way to get this done in future projects would be to take initiative and ask about those things ourselves. We could have asked for a formal lecture/tutorial about the application, which would have been useful for all the groups, and therefore would have lead to more progress for everyone. We could also have contacted Pontus (or some senior developer in future projects) ourselves to get more information instead of waiting for a structured lecture about it, and that way been able to progress faster (have a faster relative velocity) and help other groups (and save us some frustrations).

****The behavioural overview of your application (for instance through use cases, interaction diagrams or similar)****

<u>What has been:</u>


1:50

Port Calls

Search by name, IMO or MMSI number

Sotraoil
Possible N.O.R: 04/30/2018 10:00 ANCHORED

Tankskar
Possible N.O.R: 05/05/2018 09:21 UNDER WAY

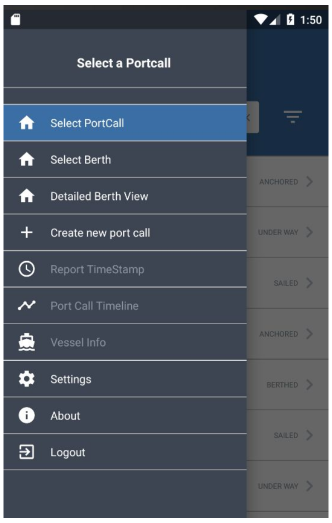
Danubia
Possible N.O.R: 05/16/2018 07:00 SAILED

Stena Alexita
Possible N.O.R: 05/16/2018 21:00 ANCHORED

Sti Finchley
Possible N.O.R: 05/18/2018 08:00 BERTHED

Sten Idun
Possible N.O.R: 05/18/2018 15:00 SAILED

Oljaren
Possible N.O.R: 05/19/2018 15:00 UNDER WAY

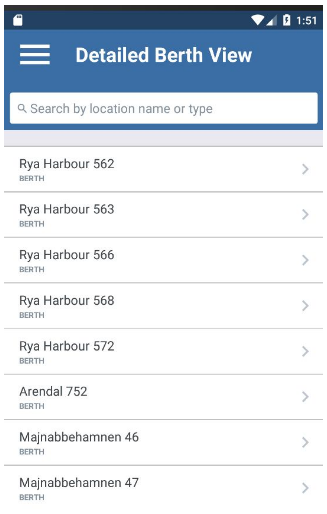

1:50

Select a Portcall

Select PortCall
Select Berth
Detailed Berth View
Create new port call
Report TimeStamp
Port Call Timeline
Vessel Info
Settings
About
Logout

As for the port call view there are no major differences in the behavioural overview. Most of the heavy lifting is done behind the scenes. The information that is shown in the subtitles is just the most relevant information, for the terminal, taken from the timeline view.

The side menu is also much reminiscent of previous version of the application. The addition is one more button called “Detailed Berth View”. This button is used in the same manner as the rest of the buttons in the side menu, which means that this takes you to a new view.


1:51

Detailed Berth View

Search by location name or type

Rya Harbour 562
BERTH

Rya Harbour 563
BERTH

Rya Harbour 566
BERTH

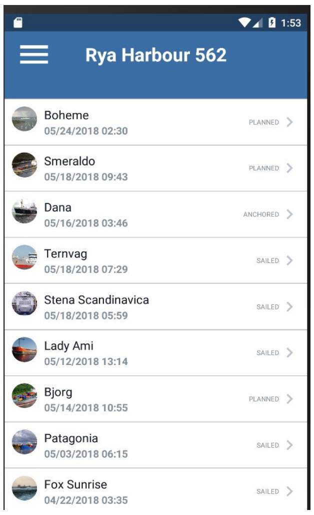
Rya Harbour 568
BERTH

Rya Harbour 572
BERTH

Arendal 752
BERTH

Majnabbehamnen 46
BERTH

Majnabbehamnen 47
BERTH


1:53

Rya Harbour 562

Boheme
05/24/2018 02:30 PLANNED

Smeraldo
05/18/2018 09:43 PLANNED

Dana
05/16/2018 03:46 ANCHORED

Ternvag
05/18/2018 07:29 SAILED

Stena Scandinavica
05/18/2018 05:59 SAILED

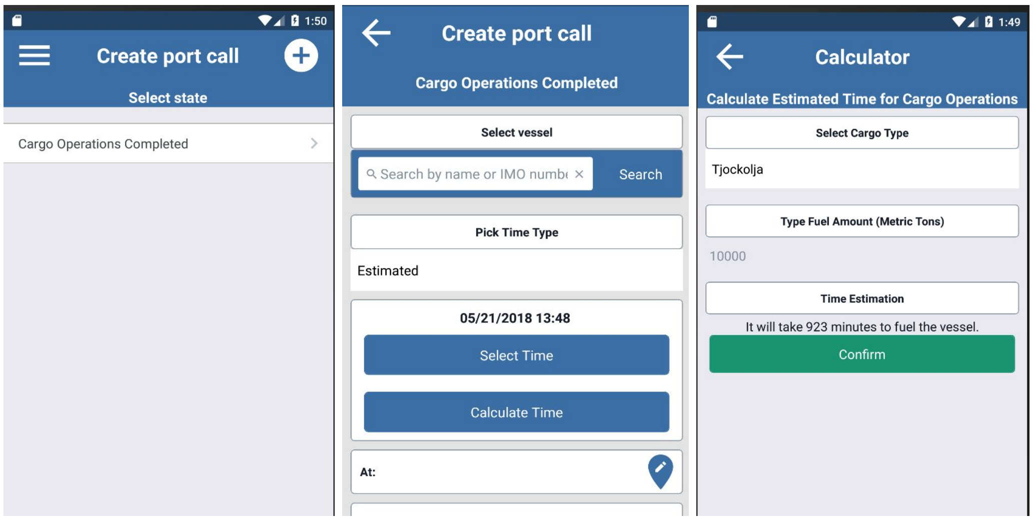
Lady Ami
05/12/2018 13:14 SAILED

Bjorg
05/14/2018 10:55 PLANNED

Patagonia
05/03/2018 06:15 SAILED

Fox Sunrise
04/22/2018 03:35 SAILED

The detailed berth view relies on showing all berths and if the user selects one of the berths - vessels assigned to that berth will show up. The reason why we made this was that the original berth view was not presenting the information for a single berth in an effective and informative manner. Therefore, there was a need for this kind of view to present the vessels. Essentially, it complements the normal berth view and together they give a good overview of vessels assigned to the specific berth.


1:50

Create port call

Select state

Cargo Operations Completed

Create port call

Cargo Operations Completed

Select vessel

Search by name or IMO numb

Search

Pick Time Type

Estimated

05/21/2018 13:48

Select Time

Calculate Time

At:

1:49

Calculator

Calculate Estimated Time for Cargo Operations

Select Cargo Type

Tjockolja

Type Fuel Amount (Metric Tons)

10000

Time Estimation

It will take 923 minutes to fuel the vessel.

Confirm

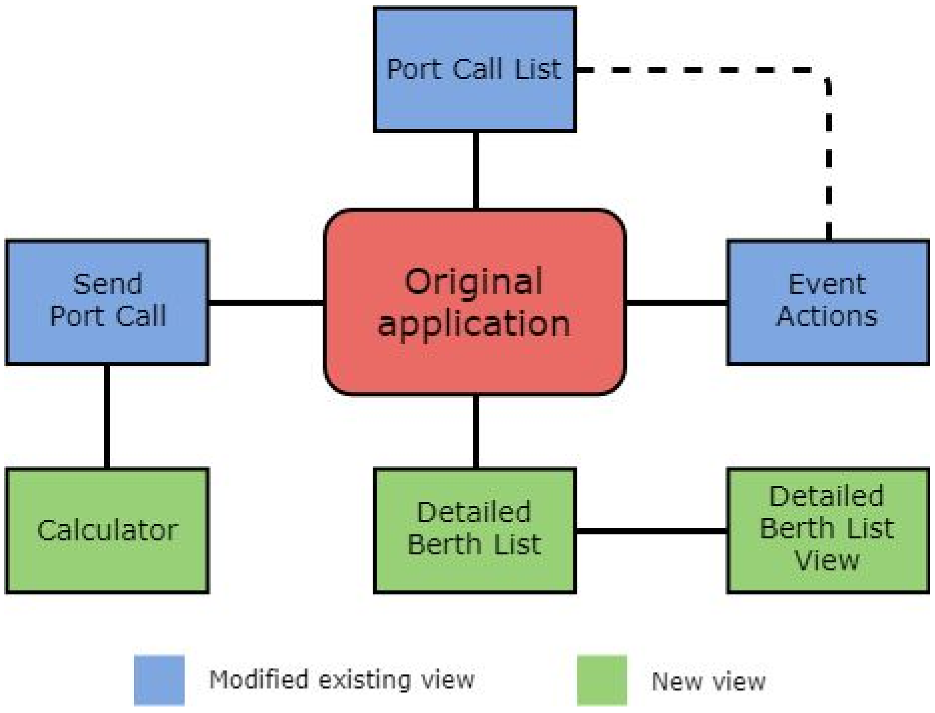
The calculator function follows the same behaviour as other parts of the application. There is an added view which you can access from the “Create port call”-window. One restriction is that the “Calculate Time”-button only exists if you choose the “Cargo Operations Completed”-state. We decided on this behaviour because it was natural since the calculator is an alternative to “Select Time”.

<u>What to do better in a future project:</u>
As this project consisted of further developing an existing app we had a basic behaviour to go by and to follow. This was both an advantage and a disadvantage since we didn’t have to design it ourselves but had to get familiar with the interface. This means that flaws in the behavioural part of the application followed into our design but on the other hand benefits followed as well. In a future project there are two possible scenarios. The first is that the software has already been partly developed as in this project. In this case it is important to follow this design so that the software is homogeneous. The other case is that the project is the start of a new software. If so, much focus must lie on designing a behaviour that is natural to the user.

<u>How to bridge the gap:</u>
In order to achieve a good design it is crucial that the group understands the project before the coding can commence. This can be done thoroughly through careful planning so that everyone follows the same design. One method may be to study similar software and to conduct some research on what behaviour that is demanded by the target group. By doing this some knowledge is gained and can be, if needed, adjusted to fit the software in development.

****The structural overview of your application (such as class diagrams, domain models or component diagrams)****

<u>What has been:</u>



The class diagram can be seen above. Most of the app is omitted since there were no changes from the original app, represented by the red rectangle. Then there are some classes that changes were made to, but the majority of the class is still intact, those are the blue rectangles with green dotted outlines. Finally, completely new classes are the green rectangles. Minor changes were also made to reducers and the navigator, mostly just one or two lines of code in certain parts, which is why they are not in the diagram above.

What to do better in a future project:

What could be useful for future projects, when working with an already existing codebase, is to establish a structural overview from the beginning, either by creating one ourselves or gettings one from the current developers. That would give us a better insight into the project and could possibly have eliminated some of the struggles we had in the beginning, saving us some frustration and allowing us to add more value. On the other hand, if there is no existing codebase and we were starting from scratch, creating and maintaining a similar structural overview would be helpful both to make sure our design is well thought-out and to be able to bring on new team members quickly.

How to bridge the gap:

To bridge the gap in future projects, we would suggest to continuously update the structural overview with the creation and design of the class. It should be fully integrated into the design process. In doing so, the overview will always be up to date and the members of the team will stay updated with how the progress goes, in context with the design. It can also be used when showing the product owners (and other stakeholders) new potential solutions, and also keep them up to date with the structure of the application. Therefore, understanding the code base is not needed for stakeholders and they can easily see how the different parts of the application is connected to one another.

****Your user stories in terms of using a standard pattern, acceptance criteria, task breakdown and effort estimation****

What has been:
Our user stories followed the “As X I would like Y so that Z”-pattern for standardization and we did our best to make sure they followed the INVEST-criteria. More information about each component of the INVEST-criteria is found below.

Independent: It was difficult to make the user stories independent since we wanted to focus on accomplishing as much as possible on our three epics (they were the highest priority according to the product owners and end users) and each initial user story consisted of coding the most basic functionality for each epic. Therefore, it was difficult to start on new valuable user stories before the first ones were completed (this took some time, more on this later in the report). Since we were four coding pairs but only had three epics, this at one point made it a bit challenging to divide tasks between us so that everyone would have something to do. The way in which we managed to tackle this was making two pairs work on the same user story but have them work on different tasks. This worked pretty well but not perfect, as some tasks were also dependent on other tasks. For the future in larger and longer projects we think this issue will be easier since we will have time to do more user stories per epic. Our hunch is that the more user stories that have been completed within an epic, the more independent the user stories will naturally become. Additionally, there would be more time to be able to prioritize smaller and more independent user stories that slightly improves the product but aren’t as important.

Negotiable: We made sure our user stories were negotiable by thinking about different alternatives. The product owners could this way easily tell us which things were most important and we could adjust the scope accordingly. An example of this was asking how interested they were in making a feature that could be customized to fit various actors needs vs making it less customizable but perhaps more clear and optimized for a certain user.

Valuable: Making every user story independently valuable was absolutely something that we focused on and felt that we succeeded with. At the same time we also tried to vertically slice user stories as possible. However seeing how long time each user story took, one could argue that our user stories were more like epics. We believe that this was due to the difficulty and not the size, and that if we had tried making them smaller then they wouldn’t have been of independent value. The size will further be discussed below.

Estimable: Our goal in regard to making our user stories estimable, was to have user stories that were small enough not be regarded as epics (thus making them easier to estimate) but also sufficiently large so that they are independently valuable. By continuously estimating our user story, we could then lean back on previous estimates that have been proven to work and with that experience, basically optimizing our estimates. For example, if a user story had an estimate of ten hours to completion but we then put ten hours of work to complete about ⅓ of the user story - the estimation was updated with three times the first estimate. By doing this, we continuously reflected upon our estimations and how they affected the progress of the work and the scope of the application. However, this was proven to be more feasible theoretically, as new problems occured for each user story and it was impossible to know how many problems would occur or how long they would take to solve.

If the project would have been longer and the rise of new problems was absent, we would think that our estimates would have improved even further over the course of a longer project. This is discussed further in the section about our three KPIs, where *estimation accuracy* is one of them.

Small:
We tried to slice our user stories as small as possible in order to keep the project moving forward and also making it easier to track our progress while still being of independent value. An example of how we tried to do this was vertically slicing user stories to limit customizability at first and then creating a second user story with customizability incorporated. However as mentioned, coding was very difficult and it was not until the last two weeks that we could do significant progress. This also led us to "secondary" user stories (with further customizability) being ignored due to lack of time or incorporated at once if it was not too difficult. As mentioned, one could argue that our user stories were more like epics due to the amount of time they took but we think this was due to the difficulty in coding and lack of prior knowledge. Making user stories smaller than they were would have resulted in sacrificing them having independant value.

Testable (and acceptance criteria): The way in which we made our user stories testable was to give them acceptance criteria (for details regarding these, see acceptance criteria question) when writing them down on our scrum board. Later when they were ready to be tested the tester could simply look at the criteria to make sure that the coders had fulfilled the goal of the user story.

Additionally we tried to break down the user stories in SMART-tasks as much as possible. Dividing user stories into tasks made it easier to track progress, make it clearer what has to be done and have multiple people/coding pairs working on the same user story. We definitely feel that we managed to make them specific, measurable, assignable and realistic but to be honest forgot to think about making them time-relatable. This perhaps could have helped us get more coordinated and complete user stories faster. For the next project this is something we will think about.

<u>What to do better in a future project:</u>
For next project, we know to not underestimate how long time coding at the beginning can take, seeing as it takes a long time to learn how to code in a new environment and there can be a lot of road blocks. Therefore, judging based on this project, the initial estimations should be scaled by a factor of three to hedge for potential problems. However, the further the along in the project we are the more accurate our estimates are, and the less we have to scale them. Hopefully for next project that is longer, we will manage to fully complete more user stories and thereby through experience get better at making estimates.

Furthermore, the concept of user stories would be helpful in both dividing work among the team members and also in keeping track of the progress. Through the course of this project, the user stories have proven to be very beneficial in keeping track of our progress and to have a predetermined scope to work against. In a future, similar project the use of user stories would feel as a natural way of structuring up the work. The INVEST-criteria was also useful in the development of user stories, as it helped us with developing reasonable user stories that on its own provided the project (and thus the product owners etc.) with value. When each user story was completed, it could be valuable for the product owners to track our progress and also seeing the value in each individual component of the application.

<u>How to bridge the gap:</u>
In order to continue to use user stories in the future, the upcoming project needs to be introduced early to user stories and connect this with the scope of the project. Thus, the members of the project will be forced to think about what the project should be like and what kind of features to include, if it is an application for instance. In doing this, the project will progress and it will be beneficial to all members to start thinking about how the project should be performed, in regard to the user stories.

****The three KPIs you use for monitoring your progress****

Communication:
<u>What has been:</u>
Overall our communication within the team has been pretty good. We have “only” had “Daily” Scrum meetings twice a week (Mondays and Wednesdays) as the course represents half of our total workload and we have also had to write our bachelor thesis. In addition, we used Facebook Messenger for communication, and to some extent Slack. The reason why we prefered Messenger over Slack was because all of us were more comfortable using it and the additional features that Slack offered weren’t enough for us to switch. Another drawback with Slack was that there was a larger risk of somebody forgetting to check the chat from time to time, as Facebook Messenger was proven to be more of a natural communication channel for each member. However, one cool feature Slack had was that it could be connected with Trello, so whenever the board was updated everybody could get an automated notification via Slack. What we mostly discussed in our group chat was reminding each other about writing their personal reports and time and place of our meetings.

More importantly though was that we had a straightforward and honest communication within the group. We made sure that everyone would feel comfortable asking any questions without feeling embarrassed and if there ever was an issue we could have a constructive discussion in person to solve it without damaging our team-spirit. As for the communication with the product owners, we felt talking with them was always very easy and helpful to get their input.

<u>What to do better in a future project:</u>
Even though facebook messenger worked fine for us, it would probably be better in the long term to start primarily using slack and to get familiar with it since it’s what most teams in the IT industry seem to use (which is understandable seeing as in longer projects it is more important to have different threads to make old messages easier to find).

Another thing is that unfortunately, it occasionally happened that one of us would forget to write the report but thanks to reminders from others the report never came in more than two days late. One time it did happen though that the product owners didn't see us and left before we had the chance to talk. Nevertheless we could still tell them about our progress and ask some questions via email.

<u>How to bridge the gap:</u>
If we do want to mainly use Slack instead of Messenger in the future, it is better to start with Slack from the get go and not with Messenger as we did. After the time where the product owners left before talking to us, we payed more attention to make sure that it didn't happen again and will continue doing so in future projects.

Estimation Accuracy:
<u>What has been:</u>
Over the course of this project, we have been working continuously with doing estimates on the time consumption of our user stories. Overall, the estimation accuracy has been hard to improve in a satisfactory manner. Even if we had small user stories and also tasks in these user stories, new problems arose on each new user story and these were problems not encountered with before. Therefore, our estimates were pretty hard to perform as the problems that occured during each user story were hard to foresee and estimate how long it would take to get rid of the problem. However, we strongly believe that in a longer project and with more experience, these problems would not be as common and therefore the estimates and the accuracy of those, would have been greatly improved.

Due to the limited time of this project and with our limited experience in developing an application (especially in JavaScript) - we found it hard to provide good accuracy in estimates. However, what we did find good with our estimates, was the fact that we reviewed and discussed them during sprint reviews. In doing so, we got an understanding of how and why problems arose and how we dealt with them. But as mentioned, we did not improve our accuracy that much from this, but what we did improve was our teamwork and our ability to find and review problems from the previous sprint. That was very beneficial for the dynamics of the team and even if it wasn't reflected in a much better accuracy, it still contributed to the project, our understanding of problems that occured in this project and how the team members cooperated in fixing these problems.

<u>What to do better in a future project:</u>
In regards to estimation accuracy, the number one thing to improve is the estimations themselves. They were suffering quite a bit from us getting struck with new problems for each sprint and thus making it hard to estimate properly. Furthermore, in more long running projects - the amount of new problems will hopefully decrease and the understanding of the already-existing problems will increase likewise. In understanding the problems and how to deal with them, along with better experience for the project as a whole, the estimations will be easier to make and the accuracy will increase.

<u>How to bridge the gap:</u>

In order to increase the estimation accuracy - one option is to work with the project longer and tackle the problems that occur, and in the long run the estimations will get better and better as our understanding for the project and its problems will increase. One option is to multiply the initial estimations in the beginning with some factor, to take unpredictable problems into consideration. As the project continues and the knowledge about these problems increase, the factor will start to decrease and continue to do so until it reaches one. When it reaches one, this means that the estimation should not be affected by unforeseeable problems. Another option is to be part of the project from the start, thus having a greater understanding of the project and therefore a better estimation accuracy. As one can tell, it is mostly about time and experience to increase the accuracy. All in all, to increase the accuracy - be patient and give it some time and the experience from making estimations will increase over time. Also, reading about the subject at hand is also important in order to understand it and thus making the estimations better.

Velocity Consistency:

<u>What has been:</u>

Overall during this project, we generally kept consistent with working 20 hours for each member of the team. Due to the fact that we kept track of our hours in a spreadsheet, the task to be consistent with our velocity was actually quite easy. This also contributed in knowing how much we could do in each sprint and indirectly help us with our estimates in what we could accomplish in each sprint, as we were consistent with our velocity. However, in sprints where we worked a bit less than 20 hours (due to the deadline of the bachelor's thesis) we could easily keep track of this with the help of the spreadsheet. We could then "pay" for our missed hours in the previous sprint and work a bit more in the following sprint. In doing this, we were consistent with the hour and thus our velocity.

<u>What to do better in a future project:</u>

In a future project, spreadsheets in keeping track of the time spent in each sprint is important and relate that to the progress in order to see the velocity. A good addition for future projects would be to write down what tasks everyone was working on in addition to the hours spent. The tracking of hours is not showing the velocity per se, but the hours need to be connected to the progress of the team and the deliverables. Another thing is to stay consistent with the hours spent each sprint in order for the velocity to be comparable between each sprint. Another way to make sure the velocity is consistent is for every team member to be on the same page and have a similar ambition level, then they will push each other when someone lags behind.

<u>How to bridge the gap:</u>

To bridge the gap, the group can begin by setting up a spreadsheet keeping the time of how much everyone spends. A social contract should enforce that this spreadsheet is kept up to date and that everyone spends their time as intended and keeping track of what their working with. Also, there should be motivation for showing up on time to group (work) meetings and doing enough work each week, by having some sort of penalty, like buying fika for everyone if a teammate shows up late or doesn't perform fast enough. The team should also be chosen after asking each potential team member about their level of ambition to contribute to the project.

KPIs in general

<u>What has been:</u>

In this project, we think that our three KPIs have been really good in keeping track of and monitor our progress during the course of this project. KPIs in general have been very useful components to the project and therefore, it is without a doubt something that will be used in future projects. It has been beneficial to the project to always have three key elements to reflect upon and compare between the sprints. In doing so, problems from earlier sprints may be recognized and also put in perspective together with the KPIs. The choice of KPIs can always be reflected upon and questioned, depending on which KPIs that have been chosen. In this project, the KPIs chosen here are quite measurable except for communication, which is pretty much immeasurable (at least in figures). On the other hand, communication is key in a project like this where everyone is new to the environment with JavaScript and so forth - therefore, without good communication it would be harder for the team to develop the application and learn from each other. Still, the presence of communication as a KPI can be questioned, and instead in a future project it might be replaced for something more measurable, like logged defects for instance. Great communication still remains as a key part of a successful project, though. As for the other two (estimation accuracy and velocity consistency), we feel that they have provided good measurements for the project and they are easy to trace back in time. Naturally, these will definitely be considered to remain as KPIs in upcoming projects.

<u>What to do better in a future project:</u>

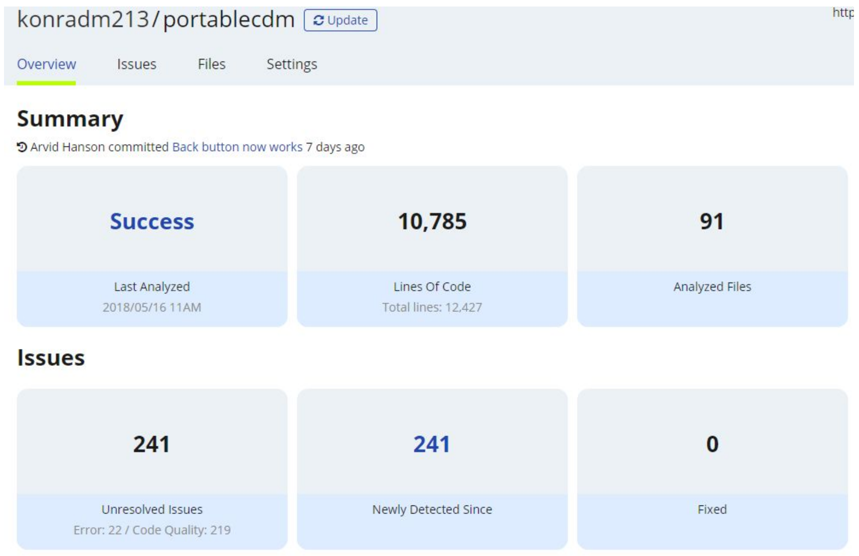
In a future project we would want KPI's that are more relevant. Especially we want measurable indexes in order to make it clear how the progress is going. By choosing the KPI's carefully these can help the group and signal if something is not working as it should and if performance is not ideal. We think it is important to have KPI's that effectively measures different aspects of a project instead of multiple indexes for one aspect. In that way they would complete each other, sort of with velocity consistency and communication.

<u>How to bridge the gap:</u>

In order to choose good KPI's the project should be well understood. Therefore, it is important to have indexes that are applicable to that project. Of course they can be changed if the reflections lead to that, but then another index should be well motivated. To test if the KPI's are appropriate the group can continuously measure these to see if it aligns with the perceived progress. All in all, in order to effectively use KPIs in the future, they need to be considered together with the scope of the project. If the project is very code heavy, then maybe logged defects together with other measurable KPIs might be preferred. This is something to discuss in the future team.

****Code quality using a tool such as Findbugs****

<u>What has been:</u>


konradm213/portablecdm
Update
Overview Issues Files Settings

Summary

Arvid Hanson committed Back button now works 7 days ago

Success	10,785	91
Last Analyzed 2018/05/16 11AM	Lines Of Code Total lines: 12,427	Analyzed Files

Issues

241	241	0
Unresolved Issues Error: 22 / Code Quality: 219	Newly Detected Since	Fixed

We made use of deepscan.io to make sure that we didn't add any issues to the project with our new code.

We made thorough use of deepscan.io during our sprints. We had no ambition to sort out the issues that were already in the code base, since this was clearly outside of our project scope, but we made sure to not add any issues by contributing our code to the project. In the end, our code contained 0 issues, but the code base we started with contained 241 . We also made sure to check each others code manually. We didn't assign code quality checking to one individual, but instead everyone was expected to check their own code.

<u>What to do better in a future project:</u>
In the future, if we will build our own application from scratch, we would make sure to eliminate any bugs from the beginning and to continually check for new bugs. Otherwise, there is the possibility that bugs will just pile up as we go and new quick fixes will be implemented in order to be able to move forward, which is not an optimal way to work for larger projects. In case we start from an existing code base, as in this case, we will make sure that the code within our scope is of acceptable quality, but take little responsibility for code outside of our scope.

Another thing we hope to do in the future is to assign a person to do code quality checks regularly. If we assign one person to check everyone's code, as well as making sure that everyone checks their own code, we can ensure a degree of redundancy that will minimize the risk of issues replicating.

<u>How to bridge the gap:</u>
If starting a project from scratch and do code quality checks often, both manually and bug checking tools like deepscan.io. If starting from an existing code base, check it for existing errors and resolve them if possible and if time allows for it. If unable to resolve them, at least make sure they don't reproduce. Make sure that everyone understand the best practice for code quality checking.

The roles you have used within the team

What has been:

Our group started out during week 1 and 2 with many uncertainties, and no clear roles created for anyone in the group. As our group gained more knowledge from course material, presentations, lectures, and the project, we began distributing clear roles in week 3. We assigned Gabriel as the scrum master, and Emil became responsible for e-mail communication with product owners, lecturers, and any other third party we had contact with. At the same time, the entire group began investigating the code of the app, and all members were still responsible for their own contribution and application of scrum in their work. Our goals during weeks 3 and 4 were to make clearer goals for the entire group if or when desired necessary. Our group soon learned that the developer team had to be split into coding pairs, in which we remained for the last four weeks of the project.

The main learning outcomes in regards to this part came in the last few weeks, when the group began to understand the app code. We realized fairly soon that it wasn't possible to simply have one role written down on paper and perform it. Both Gabriel and Emil were also a part of a coding pair, and coding pairs interacted actively to give each other assistance whenever any coding pair got stuck. In that sense, the roles of scrum master and communicator were the only ones that remained as isolated tasks. However, even they could not be performed without team input and decisions. The assignment of roles thus became more of a guideline in how to structure our team effort, so we could focus better. But in general, we learned that only true teamwork can accomplish a task with this much uncertainty and difficulty with communication, code, and objectives. We also learned that it is okay to delegate project roles at the pace that the project becomes clearer. In other words, it is okay to not assign project roles straight away, but instead assign them as the project becomes clearer and thus letting the roles develop themselves.

What to do better in a future project:

Our group learned that roles within a group are a general guideline in order to structure group work, and therefore often are supported by group members with other roles. In some cases, a role has to be accomplished by an entire group, but perhaps under the supervision of one group member. The roles in a group can also be assigned at times along the projects timespan, not necessarily in the beginning. All of these learning outcomes would be well suited for future projects.

How to bridge the gap:

By keeping an open mind, and accepting uncertainty in projects as a natural property, group roles can be delegated in a way that is constructive for the effectiveness of the project. Being flexible is vital, and aids to accept new roles upcoming throughout the project timeline, and also helping other group members accomplish their roles. Also, by being friendly, cooperative, and communicative, it is easier for a group member to know about how other group members are coming along with their roles, and aid them if necessary.

The agile practices you have used for the current sprint

What has been:

Before getting introduced to Scrum as a tool for agile development the entirety of the group had no previous knowledge regarding the subject. During our first sprints, we focused mainly on improving our knowledge about Scrum and how to implement it for agile development. Due to obligations in other courses, the team decided to meet twice a week for scrums, as opposed to doing daily scrums.

Throughout the sprints a continuous improvement of knowledge enabled a better understanding of agile practices and when/how to use them. After discussions with Jan-Philipp and Håkan regarding the reflection, the group decided to combine Scrum with some XP-practices (eXtreme Programming).

When the user stories were finalized, pair-programming, which is a part of XP, was brought in. After using pair-programming from week 4 until the end of the project a conclusion was drawn:

- Programming in pairs may not improve the productivity as in Lines of Code (LOC) per day, though it improves the quality of the code and spreads knowledge within the group.

Code review is an essential part of successful agile development. By focusing on the quality of the code rather than the quantity, less review and testing after finishing the code is necessary. When selecting programming pairs, we tried to make sure that each pairs combined coding-skill would be more or less equal in order to make sure that every pair would be able to be independently productive. Pair-programming made code review a continuous and iterative process, where both members could discuss and find solutions and one member would then produce the code on his computer. Something that was decided was to not switch pair-programming partners since everybody was satisfied with the person they were working with and user stories took a long time (and we didn't want to switch in the middle of a user story). According to literature about pair-programming, switching partners is recommended and helps spreading knowledge throughout the entire team.

Collective code ownership was practiced during the project, meaning all code is owned by the entire team so that anyone can make changes anywhere. This was enabled by a shared repository in GitHub and the use of Github Desktop.

Scrum-practices used in the project, such as sprint reviews, scrum boards and backlog refinements are discussed in other parts of this report.

<u>What to do better in a future project:</u>
For future projects, continuous integration can also be considered by using tools other than GitHub. An automatic setup can be time consuming to build and require a bit of knowledge, though it can save a lot of time. However, in a future project that is reminiscent of this - GitHub is more than sufficient in sharing the code base among the team. In future projects pair programming is welcomed, as it was a good experience in this project and worked well. As we did not change partners in this project that is something to be considered. Especially with more diverse team members, switching partners is something that would be beneficial and worth trying in a future project.

Test-driven development (TDD) is an agile practice, within XP, that wasn't implemented into our development. In TDD a small test is created before writing the code, which the code has to pass. The main reason was that it was too hard and time consuming to learn and implement. Another reason was the already existing code base. Using TDD in future projects could improve the design of the system and would enable immediate feedback on the code that's being tested.

<u>How to bridge the gap:</u>
Github is needed in order to continuously integrate the code among the team members. Other software, in the same category as Github could also be tried out. However, we think that for this type of project - Github is adequate. Furthermore, pair programming is also a beneficial technique in projects such as this one, where the pairs can benefit from one another.

With the objective of adding functionality to an existing application and code base, implementing incremental design was considered problematic. For future projects starting from scratch with a new code base, incremental design (and thus TDD) will be a lot easier to implement and would preferably be considered. By improving knowledge about Scrum and further XP-practices such as TDD (and tools for unit tests like JUnit and JMock), a beneficial implementation of new practices can be done in future projects.

****The time you have spent on the course (so keep track of your hours so you can describe the current situation)****
<u>What has been:</u>
The weekly number of hours spent working on the project has been varied a bit from sprint to sprint. This has obviously not been ideal from a scrum perspective, but you have to look at the underlying reasons in order to understand why this has been the case.

For starters, in the early phase of the course, the sprints tended to last less than the intended 20 hours. In part, this comes down to scrum being a new method of structuring a project - previously unfamiliar to the entire team. Due to this, the approach mindset had to be changed from the previous method of starting out a bit slower and then ramping up the pace near the final deadline, toward balancing the effort more evenly throughout the course. Additionally, the lack of initial understanding for the scenario made it difficult to figure out how to ideally approach the project early on. This caused a certain degree of confusion within the group regarding the scope of the project.

Furthermore, the very large amount of technical issues appearing while setting up the software (GitHub repo, Android Studio, Expo) was the root of a lot of frustration within the group during the early parts of the course. It often felt very disheartening to put down the recommended number of hours during a sprint, when new problems and errors seemingly showed up every week, and most of the time has to be spent attempting to address these instead of making any real progress on the application.

However, once these technical problems were mostly solved towards the later half of the course, the working velocity really kicked into a new gear with the coding phase. With this stage, you could really start quantifying the amount of progress made with each sprint, really increasing morale and motivation to keep putting in the hours to see results. Sticking to the desired 20 hours became a lot easier to accomplish, in addition to adding some extra working hours to the later sprints to compensate for falling slightly behind schedule in the earlier ones.

Perhaps interesting to note is the fact that the work has in the end actually been structured a bit like previous projects after all - with less progress being made in the beginning and more towards the end. Again, not ideal from a scrum perspective, but the group has attempted to stay as evenly on schedule as possible and believe ourselves to have done so to the best of our abilities. Towards the end, the difficulties instead mostly boiled down to having a lot of work in the backlog to catch up on, and only a limited amount of time available to do so. In particular, a spreadsheet of working hours during each sprint was kept, and later incorporated into the user stories on Trello. This way, we could always keep track of how much time was spent on each user story.

<u>What to do better in a future project:</u>
Reaching the recommended number of working hours was found to be rather difficult early on, due to technical problems and the large degree of freedom in choosing how to approach the project. If tasked with another project involving scrum, the main goal would be to stick to a more evenly balanced schedule throughout the entirety of the course. Another piece of takeaway knowledge from the project has been that working time could have been more studiously documented, for instance by documenting exactly which tasks have been worked on and for how long.

<u>How to bridge the gap:</u>
Flattening out the variety of working hours in between sprints is a difficult task, as the main cause for this was technical issues in the software. However, an obvious measure for countering would be to select software and code language that the team is familiar with beforehand - plus having everything installed and ready right from the start of the course or getting help from someone who knows how to set everything up. Additionally, a more precise schedule could've been useful for increasing the accuracy of the estimates during the coding phase.

****The sprint review (either in terms of outcome of the current week's exercise or meeting the product owner)****

Talking with the end user

<u>What has been:</u>
The progress during the first couple of sprints was pretty slow. This was due to a number of reasons, the first one being that our contact person at the terminal didn't have time to meet us until the fifth week. We did try talking to the product owners but it wasn't really until we had the chance to talk to the terminal workers that we could write down good and relevant

user stories that would bring some real value to both actors (we did have some user stories before but they had to change after our interview). By talking to the terminal we could get a much deeper understanding of the scenario, what their role is, how they operate on a daily basis and how we could develop something that they would find valuable.

<u>What to do better in a future project;</u>
For our next project we now know how important it is to talk to the end users as early as possible, however during this project there unfortunately wasn't really anything we could have done to get an earlier interview as they were too busy. Another thing to consider for next project is to have more contact with the end user and show them the progress continuously to get their feedback. By continuously showing our progress to the end user, we can get reviews and opinions and new ideas to present to the product owner.

<u>How to bridge the gap:</u>
As stated above, in this project there was not much that we could do about the fact that our meeting was postponed. However, in coming projects, to avoid postponement, the option to change the person you are meeting with should be considered. In doing so, the meeting could avoid to be postponed and the progress not be halted (in some ways).

Working Environment (which was the focus of our reviews for the first couple of sprints)

<u>What has been:</u>
The next reason was getting the working environment to properly work on everybody's laptop. Many in our group had a lot of difficulties to get it to work while a couple could do it faster and start helping the ones with difficulties, which was very helpful.

<u>What to do better in a future project:</u>
However one mistake we made was that in a fairly late stage we discovered some new problems, such as the importance of everybody using the exact same repository to later facilitate merging branches and the necessity of implementing git-ignore files. This caused us to have to take a few steps back, as we then had to re-do some of the working environment installation.

<u>How to bridge the gap:</u>
We think that the reason this happened was because we had not fully understood how Git works and that many files would be edited by themselves depending on whose laptop was being used if they didn't get git-ignored. Most of us had never used Git or Android Studio before but now that we have gathered some more experience we think that for next time we will be more prepared. Another thing we could do to avoid the mistake we made was to talk more with other groups that had already been able to get it to work and this way avoid some of the potential pitfalls.

Lastly, another option that could have shortened the amount of time it took to get the working environments to properly work would have been to not prioritize all team members having their own working environment. Seeing as we did end up coding in pairs this would have been alternative but looking back we think it was the right choice as there would have been other drawbacks not everyone having their own working environment, including more internal dependency, less flexibility, not being able to divide tasks within coding pairs and the risk of the one who didn't have the working environment to not be as involved (leading to that person learning less and not being able to help as much).

The next problem we faced once we did have a good product backlog approved by the product owners and everybody's laptop setup was the high learning curve related to the coding. It took some time to learn the structure of the app and how JavaScript worked before we could start doing some real progress, and on top of the various unexpected issues (such as the working environment suddenly not wanting to cooperate) appeared during the course of the project that slowed down progress. Especially in the beginning, this made it very difficult to complete entire user stories during single sprints. Since user stories and time estimates have already been discussed, it will not be discussed any further here.

Sprint review meetings and outcome

<u>What has been:</u>
Nevertheless, having internal sprint reviews within the team and with the product owners on Wednesdays was very productive. By having sprint reviews we could talk about what we had accomplished (even when it was not entire user stories), what went well, how we could help each other, what we could improve and concrete ways to do so (these are discussed a bit here and there in the report). During the meetings with the product owners, we could negotiate which user stories to prioritize, get feedback on how to tweak them as new information regarding the scenario was discovered and get feedback on the things we could demo. It was also on Wednesdays (our sprint review day) that we put time on refining the user stories that had been selected for the next sprint by assigning who would work on them, defining its acceptance criteria, thinking about the tasks that would be necessary and estimating time efforts (this was all done in our trello scrum board).

After the meeting with the end user we saw the potential of a new state called "NOR" (Notice Of Readiness). This state was something that was important for the end users to know as it gives more relevant information than the ETA. Another outcome of one of the sprint reviews was that we managed to convince the product owners that this was a useful state and therefore they have started to implement it into the official app. The NOR is also used in one of the user stories we developed.

<u>What to do better:</u>
Something negative that happened though was that one of the user stories that we had started on had to be scrapped despite it having been approved by the product owner. There were a few reasons for this, one was that it required much more work than originally estimated (because we would have needed to create a database), another was that we found an existing that function could sort of serve as a substitute and the third was because it was not a very highly prioritized user story to begin with. In the future we would like to avoid this because it wastes time.

<u>How to bridge the gap:</u>

The main way we could have avoided this would have been to think through the necessary tasks related to the user story a bit more, this way we would have realized that creating a database would have been required (which is too complicated for us) and we would have avoided wasting time on it. The positive thing was that this served as a wake-up call, after this incident we put much more focus on dividing user stories into tasks. Not only did this prevent us from doing the same mistake again, it also led to other benefits as well (these are talked about in the section about user stories).

****Best practices for using new tools and technologies (IDEs, version control, scrum boards etc.)****

Version control (and IDEs)
<u>What has been:</u>
Throughout the whole project many technology problems occured, most of which with getting exponent to start and working together with the emulator. On many occasions it took up considerable amounts of time to get all the pieces working before being able to actually start coding, which meant that the amount of value adding time was far from 100%.

Another set of problems arose with the setting up of GitHub which none of the team members had significant experience with beforehand. After all of the gitignore files were in place and branches were created everything seemed to work fine for most of the team members though, even when switching between branches. Once again, on some occasions errors popped up within the app, but only for one of the two persons in the coding pair, despite having the exact same version of the code. We are still not sure if this is due to problems with exponent or some weird syncing through Git. In the last weeks of the course we heard from Pontus that there are often errors when using exponent with windows, which could also be seen inside the group.

Not everything is negative though since we do feel that we have learned a lot about both GitHub and exponent which is something we will definitely take with us into our next projects. We also learned a new programming language, mostly foreign to the whole team, which is once again valuable for future projects. We did not use any IDE, but for editors the group was split between Sublime Text, and Atom. Both seemed to work fine. Another useful tool was GitHub Desktop, which was recommended by Arvid, because it gave a good visual overview of which files would get pushed, and the ability to easily check off those we didn't wish to push. Also the Merge function in GitHub was very helpful, and allowed us to work on many different parts of the app simultaneously. We got a merge conflict error but it was also easily handled thanks to GitHub Desktop.

<u>What to do better:</u>
Very similarly to the design of our application section, some lectures or tutorials about getting exponent to work would be great, since the errors took away from the productive coding time. In this case it is not feasible to change our hardware, from Windows to Mac, to be able to better develop with exponent just for one course. Other than that we have learned many things during the project and we are satisfied with our process of taking on new technologies, where one or two people take the charge and then explains the most important

parts to the others. This division of labor is helpful since it allows the group to work on many things in parallel and therefore be more productive in total and is something that we will most likely use in future projects.

<u>How to bridge the gap:</u>
A possible way to make taking on new tools and technologies is even more efficient is to actually designate one or two members for that task specifically (it would be their role in the team, in addition to being on the development team). They would be responsible for establishing a baseline of best practices for that new technology, and then bring the rest of the team onboard. Another way, similar to the one described in design of our application section, would be to take initiative ourselves and find out the best practices and new tools by talking to the current developers or other teams. If successful that would be a significant decrease in the time it takes to get up and running.

Scrum board:
<u>What has been:</u>
For our scrum board, what we used was a trello group. We had read some literature with guidelines on how to make and use a scrum board going into the project but actually learning by doing was very helpful and during the course of the project our scrum board evolved into something that was much better than what we started with. From the beginning we were using many different boards, one for each sprint and one for the product backlog. However after getting some feedback from Håkan, we changed it so that everything would be on the same place. This had the benefit of getting a better overview of the things that were happening without having to switch boards back and forth. This also reduced the risk of a user story getting left behind and forgotten in a board belonging to an earlier sprint. Another thing is that it allows us to easily see everything that we have accomplished, which was important for us to feel like we were making good progress.

However apart from this we got most things right from the start. Our board had five different columns, one for the backlog, one called to-do with items that were to be worked on during the next sprint, one for things in progress, one for things that are ready for testing and one with everything that is done. To make it easier to see how tasks and user stories were connected to each other we used labels to color code which epic they belonged and numbers to show which tasks belonged to which user story (3.1, 3.2, 3.3 etc). Other trello features that were helpful for us was being able to assign team members to different cards, checklists for the acceptance criteria, definition of done and tasks (sometimes we used checklists for tasks instead of seperate cards), descriptions to write down estimates and attachments to add sketches of what user stories were supposed to accomplish.

<u>What to do better:</u>
However one scrum board feature often mentioned in literature that we didn't use was a burn-down chart. The reason for this was primarily because it was hard for us to give accurate time estimates (this has already been discussed), so it would have been impossible to make a burn-down chart that actually showed accurate information. Another we could do better during the next project is for every team member to be more diligent in regards to keeping the scrum board updated. For example it sometimes happened that people forgot to

check boxes and that somebody else noticed and had to do it much later. Keeping an updated scrum board is an important part of scrum because it allows for all team members to easily get an overview over what's happening without having to hesitate if it's updated.

How to bridge the gap:
By managing to make better estimates during our next project, it would be very interesting to give burn-down charts a shot and see how it works in practice. A way in which we could improve our estimates would have been by writing down our actual time effort in trello instead of in a separate spreadsheet. This would have made the information easier to access and more central in the process. It would also have made it easier to write down the time spent on individual tasks (that are seperate cards) instead of entire entire user stories. These things would make it easier to make accurate estimates.
As for keeping the trello board updated, it's simply something that every team member must take seriously. Even if it's something that is explained to everybody, if a team member doesn't see the value in an updated scrum board, it's likely that he/she will forget. Hopefully this is something that will come naturally with more experience.

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

What has been:
A large part of the reflection is based on how we relate to the concepts brought up in literature or in lecture, some examples are the importance of INVEST criteria or the importance of estimation accuracy and velocity consistency when calculating the scope. For those reasons we will not repeat many of the things written earlier here.

In the initial weeks of the project the group tried to apply scrum and develop our sprints. However we found it difficult to fully grasp the applicability of scrum as we didn't have anything concrete to work on. Unfortunately, it was during these weeks that all lectures and exercises were. However, it was still useful to have these lectures as we have been able to look at them in hindsight and bring the knowledge into practice. The group divided the literature among ourselves in order to be effective in our working process. Later we could share what we had read and learned. For example, several members of the group read "*Scrum and XP from the Trenches*", which has helped both in understanding scrum as a whole but also in detail. As the project has unfolded we have come closer to what the literature and lectures are saying, for example by standardizing our user stories and breaking the user stories down into appropriate tasks.

Some concepts raised in the literature, like daily scrums in the form of stand up meetings, weren't directly applicable during our project. This was mainly because of the structure of the course and due to team members obligations in other courses. Besides, when we did sit down to do work, we always sat close to each other which allowed for informal meetings. This had the effect that we we're always coordinated enough to lessen the need for formal daily scrums.

Also the book “*Scrum and XP from the Trenches*” suggests that there should be 5-15 user stories per sprint which hasn’t been the case for this project. This was because of the circumstances of the project and has been elaborated on in the questions above.

<u>What to do in a future project:</u>
It is hard to say what the aim should be. Depending on the nature of our next projects it is not certain that lectures will be used. It is however important to use the perspective one has and also gain some broader perspectives related to the project. Therefore it is important to learn much along the way, both by reading up on subjects and by having a keen ear to learn from the next project environment. One thing we could have done during this project was to explicitly ask for an introduction or lecture about the application as it was hard to understand at first.

If a future project is a full-time commitment then more concepts of scrum can be used. Daily meetings will then be applicable which can help to drive the project forward. This could contribute to the user stories in order to achieve the goal of finishing 5-15 user stories per sprint.

<u>How to bridge the gap:</u>
A good way to incorporate information from lectures or other people’s thoughts is to create a reading list in the beginning of a project, in the same manner as a reading list was given to us in this project. It will allow us to get a better understanding of the theories which we can then test and keep whatever we deem useful.

It is crucial to not have a too narrow perspective, meaning not stress testing the ideas, when it comes to gathering knowledge from literature and lectures. Continuously bringing new ideas makes for a more dynamic reflection where we can improve our work process. It will allow us to cherry-pick the most useful ideas while regularly looking for new best practices. By asking others what they have learned or what they have done will help the group to perform better reflections.