FINAL TEAM REFLECTION

M/S STENA DANICA

We decided early on that we were going to make a travel planner. Our main features were that you should be able to search trips with data from both public transport and ferries, show routes on an interactive map, save trips, see the information of a specific ferry, live push notifications with live data and see a detailed view about the trip. These features were also our top priorities. Most of them we succeeded with but we couldn't manage to implement live push notifications and public transport for other areas other than Western Sweden and ferry routes other than Frederikshavn-Göteborg.

We chose to make a mobile app because it would be more convenient for the regular traveler (as everyone carries around a mobile device). Our choice of platform was Android, as making iPhone apps basically requires one to have a Mac, something most team members didn't.
Our choice of a design pattern was MvvM, we chose that one because it was the one recommended by Google and it allowed us to update the UI live which is a lot more user friendly than our users having to manually update the app.

When we did document, we used ordinary comments in the code. Our documentation is quite lacking, this too is something we wished we had done from the beginning. Each week we said we were going to do it, but the longer we waited, the harder it seemed to be to go back and do. Towards the end, we also became very focused on the actual end-product and how it would appear to users and PO's. We decided to follow the standards of Android Jetpack as it is the current standard for making Android apps and it is highly recommended by Google.

In the beginning our user stories looked more like epics than user stories, and when we tried to break them down we made them into task size instead. But as time went on we became better and better with good sized user stories that we could break down into tasks. But we often had tasks that just kept on growing as we started to work on them and we had to break them down into smaller tasks. We initially had some problems with assigning the tasks equally to team members, but these vanished pretty soon.

At the start of this project we didn't merge regularly and that led to us having a lot of merge conflicts, but as time grew we started to merge more often which led to us having fewer merge conflicts. Another problem was that we implemented the entire design model during the first sprint instead of gradually building on it throughout our sprints, because of that we couldn't work vertically (because we needed the model to be implemented before we could start coding).

As for technical documentation, we very early made UML diagrams of domain and design models. We felt this helped us during the start of the project. We were planning to constantly keep these diagrams updated, but this proved too hard in the long run (since the structure changed a lot as we worked on our tasks).

As for acceptance tests, we did have acceptance criterias on our tasks but didn't use automated testing. Instead, we tested our features manually using a test document as a reference for how the app should work. In the later stages of the project, we did start to regret this, as it became harder to test to see if everything worked and we missed a lot of bugs. We deeply regret that we lacked the discipline to use automated testing, especially as we stated we would in our social contract.

One KPI which we improved upon was merging, at the start we had a lot of merging conflicts but throughout the project the merging conflicts diminished. Our velocity also increased throughout the project, at the start we worked horizontally (e.g we had to wait for the model to be implemented), but the more tasks we finished the more vertically we were able to work. One KPI feature we've heard of but didn't use was assigning emojis/smileys to the last sprint, thus expressing our feelings on it, this is something we would like to try in future projects.

We tried to follow most of the points in our social contract, but we probably should have updated it when we realized we were not using parts of it.
The main points we didn't follow were code review (check for proper testing and documenting) and we didn't always do a proper reflection around our sprints. Not using code review punished us quite severely, especially towards the end. It introduced bugs and allowed members to merge features we hadn't agreed to implement. As we'd like to avoid these problems in future projects, our general sentiment is that code review is a mandatory part of the process. We didn't keep track of the hours spent on the project, but did at least meet around eight hours a week.

We did have a scrum master, but during the project we did not really utilize this. We got to meet and speak to the product owner during Mondays. We did have a bit of collaboration with the other groups, where our scrum master went to talk to the other groups about sharing data.

We met relatively often, three times a week. During the meetings, we mostly worked on the project and resolved any issues we had gotten stuck on. We regret not having a more general project discussion during our meetings, such as the structure of our project.

What we felt we worked the most agile on, is thinking vertically and starting small with a bit of value, and expanding features with more value during the project. We were worse with actually executing the sprints and doing reviews. The closest we came to that, was talking a bit about the state of the project when we met. We did not have "daily scrums" every day, but we did at least try to keep each other updated

on what we were currently working on. The sprints usually started good, but quite quickly turned into people working on more than their actual tasks. We tried to update our Trello board to encompass the new tasks people took upon themselves and add them after.

Seeing as no one of us had used Android Studio before, most of the first week was spent learning the platform while we sketched out how we wanted the app to look. It still came up during the project that we were new to Android Studio, and a lot of the problems and confusion we had was because of that. Git also proved to be more troublesome than we anticipated, partly due to problems with the 'rebase' command, but also due to its integration into Android Studio.
As for learning new tools and technologies in general, we believe in learning by doing. Initially, one sometimes need to follow tutorials, but pretty soon the fastest way is usually getting hands-on, complemented by googling (i.e. Stack Overflow). We can all say that we are more comfortable with Android Studio and Git now, after having worked with it for a few weeks.

In summary, we have learnt a lot about working agile and the advantages of it, by not working very agile and finding problems with our way of working. It is now, after we are done that we can easily say that it probably would have gone a lot smoother had we followed more of the agile practices. Especially having sprint reviews, where we could have improved our methods of working each week. And making a point of having real daily scrums everyday, even if it just happens online, would lead to better communication within the group. This would have let us avoid people being confused when new updates were pushed without knowing anyone was working on them.