DAT257 Nevarro Report

Chalmers University of Technology, HT 2 2022

Jesper Jansson, Henning Dahlén, Tomas Alander, Alexander Bratic, Alen Mukaca, Andin Blomgren, David Azizi

Abstract

For about 8 weeks, this team has worked together to create a web application called “Eco Travel Planner”. The main purpose has been to learn Scrum and how to work together using the Agile methodology. The team had a sprint planning meeting each Monday and a reflection meeting each Friday, with some additional stand-up meetings during the week. A new Scrum Master and a new Product Owner was chosen each week within the team to better simulate the Agile methodology. Trello was used to organize the planning, which included writing user stories, tasks, evaluating them and creating burndown charts. The group improved their way of working and got more effective at using the Agile methodology as time went on. During the first weeks, a lot of time was spent on learning the process of Agile, planning the scope of the project and its requirements, learning how to organize within the group and especially learning the different tools that that were necessary to build the web application, including React using next.js, Java Spring Boot, accessing information from Google’s API and many other techniques that were used for this project. In the end, a web application was finalized that closely resembled the original idea the group had at the start of the course. Although, some parts, such as the inclusion of airplane data and a map of the route, were not finished before the deadline. But as a whole, the group is satisfied with the finalized product and feel like they have increased their understanding and gained experience with the Agile methodology.

Answer structure

A: Reflections on the current project.
B: Thoughts regarding the next project.
A→B: Actions to be taken for the next project.

Table of Contents

Customer value and scope

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

Scope:

- ➢ Website with a travelplanner which estimates the carbon equivalent emissions of the trip.
- ➢ Website should have the capability of giving you the directions from one place to another.
- ➢ Website should be able to give you the emissions statistics of a trip.
- ➢ The different alternative transportation modes are compared with each other when it comes to relative carbon equivalent emissions.
- ➢ Personal customization of carbon emission estimation based on own vehicle.
- ➢ Comparisons of trips with a set of other climate emitting activities, like for example the emissions from consumer goods and services.
- ➢ APIn should have two routes, one that delivers the travel path for various modes of transportation and one route for in depth data about the emissions.

A: The team scope was to make a comparison app for traveling. The purpose is to show CO_2 emission data for different means of transportation.
Prioritizing to get data for the most common means of transportation such as bicycle, car and public transit. Transportation with ferry and plane was also on the agenda but was down prioritized for other more relevant data queries which lead to a more user friendly product.

Eco Travel Planner is aimed at any one who wants to contribute to a more sustainable way of transportation. It's fast and easy to compare different ways of transportations means, and the autocomplete function makes it fast and easy to use.

A functionality of giving directions, like a map (point two in the scope) was not finished before the deadline. Also the last point in the scope wasn’t quite finalized. The API only gets one distance for the searched route instead of different data for different transportation modes. A viable product could still be created without these features, although having these included would have added more customer value.

B: In the next project it would be helpful to have most of the user stories and tasks created in the beginning of the project to be able to identify the most complicated and time consuming tasks. That way it's easier to prioritize them early to make sure they don't pose a risk to the MVP by being left to the end project's deadline. This will also lead to a better overall picture of the project.

A→B: To do this we need to spend more time on planning and writing user stories in the beginning of the project. It's important for the Product Owner to evaluate which data queries are the most time consuming and complicated so that the team can get started with its user stories early in the project.

- *The success criteria for the team in terms of what you want to achieve within the project (this can include the application, but also your learning outcomes, your teamwork, or your effort.)*

A: The success criteria for this project was supposed to be creating a web application to display and calculate the emissions of different transportation alternatives, this is referenced in our definition of “minimum viable product”. The goal of our group was to get more experienced with the way of working with agile development, and at the same time learning more about the projects coding languages and frameworks, such as React, HTML, CSS and Typescript. Along with learning the concept of a separated frontend web app and a HTTP API which required us to understand http more and how to do dynamic web requests in JS.

B: For the next project, success criteria should be more defined, so that it’s clear from each group member what they expect and want to learn/achieve from the project in question. It might also be good to appoint someone to make sure the success criteria is adhered to, it can be easy to be caught up in development and forget about stakeholder involvement for example. Or focus too much on producing code and neglect education.

A → B: To achieve this there would have to be a meeting or discussion at the start of the project, where everyone defines what they want to achieve during the course of the project, and how. The success criteria should be divided into smaller parts. These parts can for example include feedback from the PO or happiness of the development team or some form of measurement of code quality.

- Your user stories in terms of using a standard pattern, acceptance criteria, task breakdown and effort estimation and how this influenced the way you worked and created value.

A: In the beginning of Nevarros' project some of the group members had a hard time understanding the difference and use of user stories and tasks. Userstories became tasks which lead to unnecessary big and complicated tasks. A quarter into the project the group started to grasp the meaning of user stories and tasks which lead to a more structured and less confusing way of implementing the working method in the project. As the project went along the group got better and better at using and structuring user stories and breaking them up in tasks. The group could spend less time on writing and evaluating tasks which lead to smoother and less time consuming meetings. Breaking big user stories into tasks also led to that the group were able to finish more tasks then before which led to more value to the stakeholders. The effort estimation took a long time in the beginning and wasn’t very accurate, but got better and more efficient when the group gained experience.

B: Make sure the whole group understands the concept of the agile working method such as writing user stories and tasks. If the group makes most of the user stories in the beginning of the project it will also help to get a better understanding of the project as a whole. It's also important to make a clear connection from the task to the user story in order to make sure it's relevant, and vice versa to make sure they are specific.

A→B: Invest the time at the start of the project to write clear and easy to understand user stories, make sure that the whole team understands the concept of user stories and break them into different tasks. Make sure that when writing user stories, they focus on the customer's viewpoint and not on the technical implementation. The user stories could also be split into a task outside the sprint planning meetings to make these more concise and every team member might not be needed for the job.

- *Your acceptance tests, such as how they were performed, with whom, and which value they provided for you and the other stakeholders.*

A: Throughout the project, no acceptance/unit tests were created. After sprint 2 a requirement for any code to be committed was being able to build and lint was added to the DoD. This was added after it was noticed that code was being committed that couldn't be compiled. We never had any tasks related to making tests for the code.

B: For a project of similar scale the approach of manual testing and making sure the project can compile is most likely sufficient. However for larger projects making unittest of implemented features should be added to the DoD and no code should be completely without tests. There should however also be some overarching smoke test manual/automatic to make sure that the different parts work together. After that it depends a lot on the scale of the project. Finally, it's important that tests are being run and any failing tests are reported to the team.

A→B: One or several stories would need to be added for the creation of a smoke test and also tasks to keep it up to date. Furthermore in the case that creating unit tests is added to the DoD additional consideration while evaluating points for the tasks would have to be taken since it would require more time.

- *The three KPIs you use for monitoring your progress and how you use them to improve your process.*

A: For this project we used velocity, burndown charts and PO feedback. For the velocity we looked at the burndown chart for the last sprint and also the time dedicated to the development each week and other factors such as educational cost. This was then applied to the next week's velocity to adjust for any what we needed. This was done on a team basis so every person picked from the same velocity pool. For the POs feedback, it was mostly compliments since the PO was happy most of the time. And the POs feedback didn't really get materialized other than for the next week when the PO themselves prioritized the sprint backlog according to what they experienced their wants/needs to be.

B: The velocity could be improved,. since every person has different experiences in the field or different situations in life it might be good to have some sort of personal target to aim for.

This could make it more equal so the more experienced people don't slow down when they start to hit their velocity if it would be distributed equally and reduce the feeling for less experienced people that they might not deliver enough even though it might not be reasonable for them to do so. Also the POs feedback should have been more concrete in a way that lets the team look back and learn from previous sprint one what was done wrong or more importantly right. It's also important to do this over the time of the whole project, since the reduction in happiness might be slow or the causation might have started long ago, for example technical debt. To ensure the team is on route the SM could show the burndown chart more often to incentivize the team to keep on track. This would be easier with a stable SM that is more comfortable in their responsibilities.

A→B: To get a more personal target to aim for one might create a more granular velocity chart that can be sorted for personal targets as well or just make each team member keep track of their own "score". It might be valuable to consider if this should be public, since this might lead into a feeling of competition. For the POs feedback a form to be filled out at the end of every sprint covering key questions could be made. This might be from the PO or the end users, this could then be used to reflect upon in the sprint reviews.

Social contract and effort

- *Social contract i.e, the rules that define how you work together as a team, how it influenced your work, and how it evolved during the project (this means, of course, you should create one in the first week and continuously update it when the need arrives.)*

The social contract contained the following:

- ➢ No meetings on weekends rather have sprint planning monday 8 o'clock, sprint retrospective friday 13 o'clock and having stand-up meetings at 12 for tuesday, wednesday & thursday.
- ➢ If a person cannot join the meeting in person then the team should allow remote connections for a hybrid meeting.
- ➢ Listen to other people's ideas and come with constructive criticism.
- ➢ Ask for help if you're stuck and be helpful when someone is asking for assistance.

A: There are two primary functions to the social contract when it was created which was assisting others and the handling of the meetings. However the stand-up meetings were not originally included in the social contract until sprint 2 where we added wednesday lunch and then expanded to tuesdays and thursday in sprint 4. Due to the additional meetings more team members can get tips when they're stuck. However since the routines for the stand-up meetings were established late it resulted in many team members missing the meetings due to lack of routine unlike the sprint planning and retrospective.

B: In order to improve for the next project the social contract should include the standup meetings from the get go and increase the planned time for the sprint planning. In addition, include some definition of done criteria for merging into the main branch such as the requirement for code reviews and making sure the change allows the project to build successfully in order to reduce the amount of bugs that enter main. Moreover a minimum hour requirement per sprint could be included as well.

A→B: Complete the social contract as much as possible in sprint 1 to lower the risk of needing edits part way through. With a robust contract early on allows the team to establish routines for meetings and for the merge criteria regarding the main branch.

- *The time you have spent on the course and how it relates to what you delivered (so keep track of your hours so you can describe the current situation.)*

A: The productivity varied between sprints and the different team members due to the difference in experience regarding web development. This led to many team members having to spend the early sprints reading documentation and learning the new frameworks which increased the spent time per task significantly relative to the amount of points given to the tasks. However as time progressed our efficiency and velocity improved due to us finally learning scrum as well as the frameworks we were using. Resulting in the time spent relative to the task's points to be quite reduced in later sprints compared to the early sprints. Some people also co-coded such as Jesper and Andin when working on the page template in beginning or Tomas and Henning for the autocomplete feature towards the end. But assigning tasks to two people with the intention of co-coding usually resulted in less progress due to difficulties with our different schedules so we gave up on co-coding mostly. Some tasks were also harder than anticipated such as adding an embed of google maps and configuring it took too much time which resulted in it being scrapped due to time constraints.

B: In order to have the velocity not getting lowered too much initially when starting the next project we need to better gauge each others' abilities. Since starting with many new frameworks/languages will require a lot of time to learn so on the next project we should reduce the amount of new technologies and sticking at least partially to those we already know could speed up development. Additionally, assigning a task to two people should be avoided and instead having co-coding be done in individual sessions when help is needed rather than having a task depend on co-coding. Moreover, we should assist people or maybe even reassign tasks when someone gets stuck and cannot progress on a task for a certain amount of time.

A→B: For the next project we should evaluate each other's skill sets and choose a language/framework that is most familiar. In addition, prevent assigning two people to a single task and take more immediate action when a task doesn't progress for a while. Time-boxing can be used to make sure people know when to ask for help and don't get stuck for too long.

Design decisions and product structure

- *How your design decisions (e.g., choice of APIs, architecture patterns, behaviour) support customer value.*

A: We didn’t take our time in the choice of our frameworks and instead focused on getting started with development. So we quickly picked react using next js for hosting, this gave us developers a chance to learn relevant new technologies but at a probable cost to the customer value since the teams experience in react was close to zero. This was also something we didn’t have time to backtrack on in such a short project. But considering in this case that the customer was fictional it might be a win out of an utilitarianism view-point.

The application's simplicity didn’t necessitate the use of any patterns so a more direct approach was chosen to specify the behavior of each page load. To simplify the development session storage was used for data storage between pages, so every page or page part that had data that needed to be persistent using the site read from session storage on load and written on every update back to session storage.

B: A lot of development weight could be removed by more carefully evaluating the choice of technologies considering the customers needs. Other platforms could have been better considering the scale of this application and the team. Although the pros of the chosen framework has given a robust platform to scale into a bigger application. The same argument could be made for the patterns, for a project of this scale the chosen approach worked well but this wouldn’t scale nicely.

A→B: To better evaluate the chosen technologies the first sprint could be dedicated to analyzing possibilities and weighing different options against each other inorder to make a more grounded choice both for the api and architectural patterns.

- *Which technical documentation you use and why (e.g. use cases, interaction diagrams, class diagrams, domain models or component diagrams, text documents.)*

A: A mockup, a scope and a business model canvas were created at the start of the project. Many resources and links were shared in the beginning to help everyone with learning the skills that would be necessary for the project, like react documentation, java spring boot documentation and links for learning http, css et.c. But no other technical documentation, like class diagrams or interaction diagrams, were created by the group.

B: Having some diagrams of how the data should flow would have been helpful in the beginning. When organizing what needs to be done, how many should do it and what priority a task has, it would be helpful with for example use cases or a domain model. The purpose of the back-end might have been more clear and the gathering of data from other APIs might have been more effective if we had planned ahead more with use cases, interaction diagrams etc. For example, it might have been possible to make the application get all the different distances for the different transportation modes (car, transits, plane, airplane, walking, bike), since they differ. Instead, it was decided to only use the car distance. But considering the time- and skill limitations on the team, we feel that this was a reasonable decision for this project.

A→B: Make tasks early in the project that involve technical documentation. For example, we never had "create a mockup" as a task in a sprint because we hadn't really understood or gotten comfortable with the Agile methodology yet. But in a future project, we would most likely have many tasks involving technical documentation early on, also known as planning sprints. Especially the mockup in our first sprint, but also some kind of diagram with early ideas of how the code should be constructed. In the future, we will also create tasks in later sprints to update the technical documentation when necessary.

- *How you use and update your documentation throughout the sprints.*

A: Not too much documentation was written during the project. It was brought up during some of the sprints that documentation was needed for some of the features that were either complicated or were used by the team indirectly through API calls or function calls. These were rarely followed up on though, most likely due to the fact that no points were allocated towards documentation. This persisted throughout the project and some parts of the project went completely undocumented.

B: In a similarly sized project there would only need to be a bit more documentation related to how some solutions have been implemented and how some features should be used. For a larger project however, more documentation would most likely be necessary.

A→B: The main thing that would need to be changed to get to a better position regarding documentation is to allocate time for it in the shape of points. Furthermore it would be important that there's some sort of plan in the DoD of what should be documented so that everyone in the team is on the same page.

- *How you ensure code quality and enforce coding standards.*

A: At the start of the project we suggested having comments that explain the code we had written, making it clear to understand and easy to add upon by the others in the group. This was only mentioned once in a meeting and not in the definition of done (DoD) or social contract, although we did create a user story in our Trello that said "document code". In the end, not much code had comments explaining it. At our third team reflection meeting, we decided to add a build requirement to our DoD. It says that a branch/task needs to be successfully built with "yarn build" (with linting) and without errors before creating a pull request. We have also mentioned (in both team reflection 4 and 5) that a pull request needs to be checked by another group member. Although, this has not been strictly enforced and a code review wasn't required.

B: For any future projects, we might consider adding some of the above mentioned requirements to the DoD or the social contract. For example, towards the end, our comparison page had a lot of code that had been written by many different group members at different stages of the project and it depended on many other functions or other components that had been created. Having comments in our code might have helped some group members when they received tasks that involved other group members' work. On the other hand, we had so many meetings with opportunities to ask each other about our work (part of our social contract) that this wasn't too big of a problem. In future projects, it might give value for the product in the form of it being easier for another team or other partners to build upon the project. We should also have the code reviewed by other group members to make sure it's understandable.

A→B: Add code review and maybe some guideline of how to refactor or polish before committing to the DoD. Make sure building and linting without errors is in the DoD from the start. There is a Github setting for enforcing the requirement of code reviews before a pull request is able to merge at all.

Application off scrum

- *The roles you have used within the team and their impact on your work.*

A: The assigned team roles of this particular application of scrum were not static. The Product Owner and Scrum Master of the project changed with every sprint. This made every team member engaged with the scrum mode of operations since everyone got to try each role. Other than leading the sprint meetings the Scrum Master also booked the meeting room. All other group members did not have any particular role other than developing the project.

B: Since the team is now experienced, switching the roles are no longer necessary. Having a stable role is better for the development process so to save the time during the planning otherwise spent on choosing and adapting to a new PO and SM.

A→B: This will be done by electing a PO and SM at the first sprint and permanently giving them that role. The product owner should be the group member with the greatest project vision and the scrum master should be the group member with the most experience of the scrum method.

- *The agile practices you have used and their impact on your work.*

A: Agile practices such as sprint planning-, sprint retrospective- and stand up meetings were utilized during the course of this project. Each sprint lasted one week with the sprint planning in early monday and the sprint retrospective in late friday. The stand up meetings were later introduced in the project and these occurred during lunch time everyday in between the sprint meetings. Planning estimations were also utilized during the creation of tasks.

B: A practical and robust way to align the vision of a product to every team member should be created. This will minimize the time delay every team has from the time the project starts to the moment actual work is being done to it.

A→B: To begin we could use planning sprints where we could make a complete mockup and product specification to make it easier for the team to have a shared vision. This can be done or reviewed by the PO since it is the PO that is the closest to the stakeholders. Also more time could be spent on finding more tasks in the beginning that aren’t dependent, this would allow people to work more independently early on. A further way to cement the unanimity of the team members goal is to make everybody affirm to the product specification so that the entire group is on the same page. These measures are for making sure everyone in the team has the same end product in mind.

- *The sprint review and how it relates to your scope and customer value (Did you have a PO, if yes, who?, if no, how did you carry out the review? Did the review result in a re-prioritisation of user stories? How did the reviews relate to your DoD? Did the feedback change your way of working?)*

A: We've had a change of PO each week, changing at the start of every new sprint. The PO individually reprioritized the product backlog and commented regarding scope changes before each sprint start where the team then could pick from the top into the sprint backlog taking any dependencies into consideration. At the start of the project what the role of PO meant wasn't clear to the group, most didn't have an understanding of why there was a PO. The projects DoD changed during one of the sprint reviews due to problems in the main branch in git. Feedback from the sprints resulted in more coding practice and a little less studying of documentation.

B: In future projects there would be an advantage to set a permanent PO and Scrum Master for the whole project instead of cycling through them. Have better defined epics, user stories, subtasks to the user stories, other tasks, DoD. Add a way to analyze the work and find ways to improve the productivity through the sprint retrospective.

A→B: When picking a PO, pick someone who has the best vision in relation to stakeholders. And make sure they are comfortable to be PO during the entire project. To improve productivity, make sure your Scrum Master or someone is making sure questions get asked inorder to review and find improvements (i.e. a working template, an experienced person.) There could also be a way to divide people into roles with the intention of optimizing their productivity through putting them in their most efficient position, taking in consideration knowledge is not lost in case of someone leaving their post. The sprint review should be used for this. The user stories and Epics should be written mainly by the PO, and then they should be evaluated by the development team and broken down into smaller tasks corresponding to the user stories.

- *Best practices for learning and using new tools and technologies (IDEs, version control, scrum boards etc.; do not only describe which tools you used but focus on how you developed the expertise to use them.)*

A: The technologies and tools used for the project were Java with Spring Boot as the API and HTML, CSS and Typescript with React using Material UI and NextJS. Git was used for version control, Trello for the agile board. Then everyone used slightly different IDEs such as VS Code, IntelliJ and notepad++. It was expected that everyone would have to learn a lot during the project but it later became clear that the project was too complicated for the time available and the team's experience. During the creation of the project a number of documentations for the technologies was used, learning was then mainly done through reading these or other online sources. There was some intention to do pair programming but this quickly fizzled out due to all the team members taking different courses resulting in no real overlapping time. After a few sprints the team started utilizing standup meetings in the middle of the week using voip which resulted in a lot of useful learning opportunities when someone asked for help.

B: The project wouldn't require technologies that everyone has experience using but at the same time it's useful if most people don't need to learn everything from scratch. So the project would use mostly technologies that some people know decently but also with a certain amount of opportunity to learn, that way it becomes easier to ask questions to the team when you get stuck. It would also be good with some collaborative learning sessions.

A -> B: When choosing the technologies to use in the project, keep in mind the knowledge levels within the team and don't stray too far from those. If that means that a certain project is too complex then maybe consider something else or reduce the scope. Do some sort of daily standup meeting straight from the start of the project, especially important to have in the beginning since people may need more help at that point.

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

A: During the time of our project the team didn't feel like they stumbled upon a lot of literature to use that would guide the reflections in the right direction. Most of the reflections came either from an individual member's thoughts, the instructions on canvas or from feedback during the weekly supervision.

B: In future projects there is definitely an advantage to using pre-set reflection points, not spending as much time trying to talk towards the right thought of reflection but rather start at a good point and work towards a better one.

A → B: First of all, in hindsight it's been very clear that there could have been more reflections on possible solutions when facing a challenge trying to implement something. The example from our project would be the same map task that we had for 3 weeks without completion, there was never even a talk about changing what api to use, which in hindsight could be improved. There has also been points where there was sitting around wondering what the next point of reflection would be, if faced with this there should be something to fall back on and make use of. An example would be the points that Dr. Becker brought up during his guest lecture, when an open source resource stopped being maintained and they ended up choosing to rewrite the project.