Final report

Sony Walkman WM-FX199

Alex Phu, Elin Fredriksson, Gabriel Brattgård, Hedy Pettersson, Ludvig Holländer, Stina Haglund, Yenan Wang

Customer Value and Scope

Topic	A (Now)	B (Future)	A → B (How)
Chosen scope	Invoice handling - Maybe a bit too small - Good level of how much we had to learn - The information from PO of what tasks PO was valued the highest was a bit vague	- Make better estimates of how much we are able to accomplish - would like to have a clear picture of the priorities of the tasks	- It's easier to make good estimations with more experience. - The same goes for pushing information from PO, plus if we know what we want from PO it's easier to get that information from PO.
Success criteria	Developing our programming and project management skills. Teamwork throughout the course and completing the MVP.	Delivering the product on time. Have a good workflow, teamwork and sustainable working hours.	Have an open discussion with the whole regarding these success criteria mentioned in **”B”**. This can be attained through a good project leader who encourages these things.
User stories	User stories and acceptance criteria were created by PO. We did task	Develop user stories and acceptance criteria together with the PO instead of	Experiences like the ones we have had in this course where it was from time to

	breakdown and estimations.	having them given to us. Could be a good learning process.	time difficult to know what the PO wanted from us.
Acceptance tests	Showing the PO a demo and then he gives feedback on what was demonstrated.	Continue with the live demo with the PO to be able to get instant feedback	Asking the PO more precise questions to get better guidelines for the following sprint
KPI	Used to measure the performance as a team. Answering anonymously and independently	Use them even more and directly in the beginning of the project. Collect more data to be able to catch deviations.	Defining KPIs in the beginning of the project and sticking to them.

Chosen scope:

Our PO laid out three options for us to focus on: accounting, api and security. We chose the accounting part of the application. A few weeks in we had to further limit the scope to only an invoice feature.

Success criteria

A:

At first to pass the course and learn new things in programming and project management. This developed to include also wanting to complete our product for our PO. It was very important that our teamwork worked throughout the course.

B:

Of course to deliver the demanded product to the customer, on time and how the customer wants it. A good work flow, great teamwork, together with sustainable working hours would be considered a great success for a future project. Would probably be very nice to have some smaller milestones on the way towards a complete product since this can increase morale in the team a lot. It feels a lot better when you feel that you make progress.

A->B:

To reach this we need the whole team on board, meaning that everyone in the team wants the aforementioned success criteria. This can be produced by a good project leader who encourages these things and rewards when they are achieved.

User stories

A:

Our user stories were created by our PO initially and since he had a lot of experience with the Scrum-framework the user stories followed a standard pattern alongside acceptance criteria. This ensured that the user stories brought value to the PO, since he was the one who wrote them. The task-breakdowns however, were up to us to make. During planning meetings with the PO, we discussed the user stories we were planning to work on during the next sprint. We did this by extracting information regarding exactly what value he wants this user story to bring and how it should be implemented. This was very useful during task-breakdowns. Effort estimation was difficult in the beginning of the project, since we had next to no experience with the things we were working on. This changed during the course of the project and in the latest sprints they were quite accurate.

B:

In a future project it would be a nice experience to develop user stories and acceptance criteria together with the PO instead of being handed them. This could be a good learning process for the whole team, since you through these discussions with the PO get a good picture of what he wants the project to result in.

A->B:

To get to this, one would need to work on projects until you encounter this situation. Every project you work on brings you new experiences and insights that make you a better and better teammate to bring to a project which should result in more and more effective development of user stories and acceptance criteria together with the PO.

Acceptance tests

A:

Our acceptance tests consisted of us showing the PO what we had worked on in a sprint and him giving us feedback on that demo. This feedback ranged from “just change that small part and you’re done” to more complex feedback which resulted in a discussion about what value the PO expects from that user story etc. Overall it worked very well but it would have been even better if we took this opportunity to get definite answers from the PO on exactly how he wants this user story to be delivered in the project.

B:

In a future project, we would probably perform the demos in a similar fashion to how we did it in this project. It was efficient and the PO could ask questions live during the demo. However we would in a future project ask more questions to the PO on what was wrong, how we should fix it and maybe “is there something else you think of?” to really get a straight answer from the PO on how to improve our code if something was rejected etc.

A->B:

To reach this, we just needed this course to realise our mistakes from the first demos with the PO. In a similar future project we will all have in mind that we need to get the information from our PO and we need to ask the right questions and lots of them. So all we need to reach this was probably this course.

KPI

A:

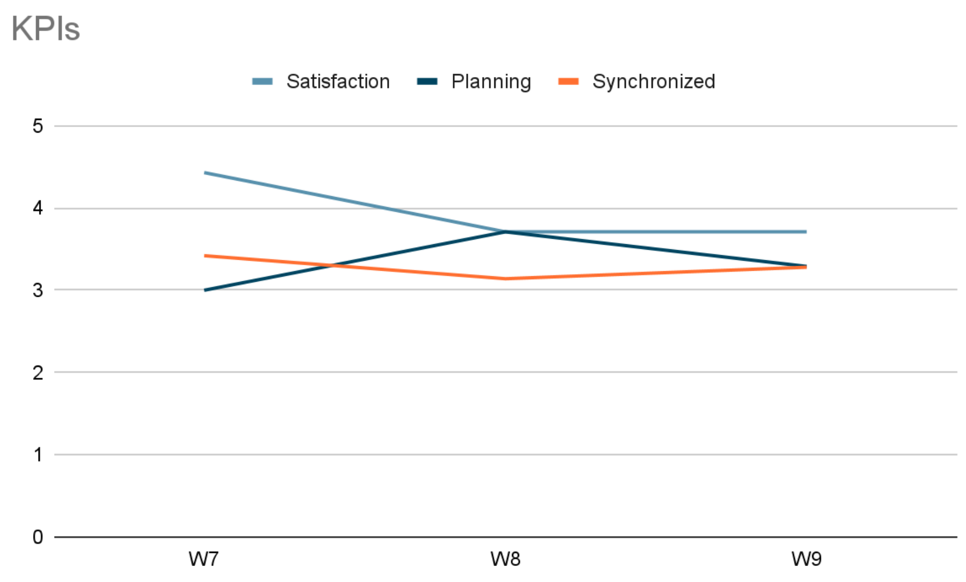
We only recently started using our KPIs to measure our performance as a team. When we used them, we first answered independently and anonymously. When everyone had answered, we discussed the results and tried to reach a conclusion if something needed to change or if we should keep up the good work.

B:

In a future project, it would be nice to use the KPIs from day one to really get a lot of data to compare each sprint with. This is very valuable for a project leader who can then detect if there are some problems boiling in the team or if everything is going great. It would also be nice to maybe have some professionals make very useful KPIs.

A->B:

To reach this, it's really about using the KPIs every week and also REALLY using them, in other words look at the data and discuss it among the team to actually get something out of it. It would as said before be very useful if the actual KPIs are good so that they bring good information regarding the teams performance.



Social contract and effort

Topic	A (Now)	B (Future)	A → B (How)
Communication	Used Discord for everything.	Same	Nothing
Teamwork	Two frontend teams and one backend team. Members switched teams occasionally.	Same structure but without the team members switching to different teams in order to improve the efficiency of the work.	Remove the team members right to switch teams.
Meeting	Everyone logs the meetings without any designated role.	Shouldn't have to worry about who's supposed to document in the log.	Assign someone as the secretary.
Social contract	We created a social contract but failed to make much use of it.	The social contract should represent the team's ideal way of working.	The social contract is consistently updated to either help or match our current situation.
Time estimation	Estimated tasks with "small", "medium", or "large".	Be able to deliver what we have promised.	Make estimates from the beginning and guess the time even if we lack the knowledge. Don't underestimate tasks that require integrating new frameworks.
Effort	Effort has been rather high since we are inexperienced with the frameworks used in the project.	We should be able to dive into the project despite the lack of knowledge without spending too much time.	This requires a helping hand from the more experienced members and pair-programming. If there are no experienced members, GG.

Communication

We decided that we will Discord for our meetings and most of our communication. However, we had also used facebook messenger at the beginning to communicate. This changed after a few weeks when everyone got used to using Discord. Discord was chosen partly because it is easy to set up meetings since you don't need to send out any link or start a meeting, you only need to join a channel.

Teamwork

When planning our project early in the course, we decided that we should work in 3 smaller teams with IT-students and I-students evenly distributed throughout the teams. We reasoned that if there are too many people working on the same thing together, we would lose efficiency and encounter the problem known as “too many cooks”. Instead we would have smaller teams and plan work during the sprints within the teams independently and instead go to another team if we had any questions regarding the code or the user story.

In the social contract, we wrote that the IT-students are in charge of programming while the I-students are mostly there for discussions about functionalities and creativity. This changed during the course where it now has been both IT-students and I-students that code and both discuss functionality and creativity, but we still go to other teams if we have any questions regarding code or user stories etc.

We also rotate our roles about every 2 sprints so that everyone understands the underlying application structure. What areas a person had worked with has throughout the course been a choice for that individual for the most part, be it frontend, backend or GUI.

In this course we focused more on the process rather than the product. For instance, most of the team members tried out both backend and frontend which slowed down the overall process since that member would then have to be introduced to the previous code base. So in future projects it would be far more efficient if the team members didn’t switch sub-teams.

Meeting

We tracked all of our meetings in a log, where we together wrote down what has been said during the meeting. In the beginning we tried to assign someone to write in the log during each meeting, but immediately after the first few meetings we stopped doing that. Instead pretty much everyone was in the log document during the meetings and wrote down relevant information from each meeting. This has been very useful since you can always go to the log and check, for instance, what the PO said about the latest demo and what feedback he gave.

In future projects, it would be more viable if we had a designated secretary for each meeting instead of not having one. Otherwise there will always be a confusion about who is supposed to be writing and we will not have the most effective meetings possible if everyone is distracted by focusing too much on the log.

Social contract

During the course, we as a team have not used the social contract that often. The reason for this is partly because we together decided what should be in the social contract, so everyone already knew what we decided in it. For that reason, there was not any need to refer to the social contract if there was any misconduct.

We did update it a few times during the course, to add what changes we made in our way of work. This can be seen in the social contract where we have timestamped updates on the social contract. However, these changes in the social contract did not change how we used the social contract. The same thing still remained, since we together had discussions about the changes, everyone already knew what they meant so there was no need to refer to the social contract during any time of the course.

This lackluster use of the social contract is mostly a consequence of our group being good at having discussions, reaching a conclusion and then adapting our work accordingly. No one on our teams has tried to avoid work or in any way been an obstacle for the rest of the team, so there has not been any need of using the social contract regularly. However, we believe that everyone has still followed the social contract subconsciously because no one on our team wants to let down the rest of the team (by skipping out on meetings or planned work with smaller teams etc.).

We started by writing a social contract and everyone signed it, meaning that we all agree with our social contract. This social contract did not include narrowed down definitions of how we should work, it was quite general. During the course it was improved to reflect how we work and how we think we should work.

In a future project, a social contract should represent what the team thinks is the optimal way to work and should have more impact on the team's actual way of working. This contract should include every aspect of the project and everyone on the team should follow it. If you have a good social contract, you will always know if your team is doing something right or wrong. To get to this point, the team will need to update the social contract frequently to include new things that the team has encountered that’s not included in the social contract. There has to be a discussion with the whole team about what should be in the social contract, since the contract is for the whole team, so a meeting before every sprint could be useful since everyone in the team usually attends these meetings.

Time estimation

In the beginning we did not keep track of our working time nor did we estimate our tasks and we discovered many problems because of that. For instance, at the first working week we did not manage to deliver what had been promised because we miscalculated the time required. So soon after the first few weeks, we started estimating our tasks. But since we were all inexperienced with our tasks it would be difficult to give an exact number. Therefore, to keep it simple, we went with the “small, medium and large” estimate criterias, where “small” means a few hours, “medium” means a few days and “large” means a whole sprint.

Those simple estimates turned out to be fairly useful. There were not as many time-estimate related problems as before and each sprint, we actually managed to finish all user stories in the sprint backlog.

The lesson has been learned about the time estimation and all of us have gained some experience with working on a full stack application. Therefore, in a future project we, hopefully, can start immediately with time estimation and provide a consistent delivery throughout the sprint.

Effort

Throughout the course we encountered many new areas. We were inexperienced with the Scrum-framework, and had next to no experience with the frameworks used for the frontend and backend. To be able to deliver what our PO wanted, we needed to learn a lot more than what we knew. So a lot of time was spent on learning these blank areas, and also to get to know how we work as a group. This means that not much was delivered in relation to how much time we spent in a sprint. This changed radically throughout the project. When everyone started to become more familiar with their working environments and the Scrum-framework, we delivered much more, without spending more time than before. In other words, we have spent around the same time on the course every week, but our efficiency has increased a lot.

In a future project, it would be optimal if everyone on the team was already familiar with the scrum-framework as well as all the working environments in the beginning of that project. This means that you can “get to work” right away. A lot of time was spent on learning all these areas, so if you can allocate that time to discussions regarding deliveries and sprint planning etc. it would be much more efficient.

There was no real way of getting to this point in this course, but we can speed up the learning process by doing pair-programming. This is one of the most efficient ways to get into a project and learn a new working environment or simply produce a higher quality result, since two heads are better than one and it is very useful to have someone to discuss code with. Knowledge about the Scrum-framework is difficult to have unless you have used it before. Therefore it is also difficult to say how one would reach good knowledge about the framework if you have not been in a project like this one earlier. It is however enough if one in the team has experience with using the framework. That person can educate the rest of the group quite fast and lead the team in the right direction from the start.

Design decisions and product structure

Design decisions and customer value

Topic	A (now)	B (future)	A → B (how)
Gold Plating	Talking about gold plating and making some initial mistakes. → Delivering an MVP	Always keep the MVP in mind and never go beyond it	Just knowing what gold plating is and that it should be avoided → good tool to actually avoid it
Figma	Making a basic prototype just to include the features → Using more features and implementing Figma components in react	Confidently making advanced prototypes in the beginning of every project No/small learning curve	Just use the knowledge we acquired during this first project and build on that through “learning by doing”
ReactJS (Front end)	Didn’t have any experience → Made a functioning website and hosted it on Netlify	Build an even more advanced React site	Starting the project with a better knowledge and more experience increases the amount of tasks being able to complete
Bootstrap	The majority of the group had no experience → Implemented some bootstrap components, like button, form, etc.	Using more Bootstrap components to save time and to make the design more consistent	Look around and search for components already implemented. Knowing it exists is a good way to start using it
SpringBoot (Back end)	Did not know anything → Knowing enough to be able to deliver an MVP	Keep developing and build more knowledge to be able to implement more suitable databases faster	Learning by doing and help from PO
Axios	Did not know anything → Being able to	Faster connection and being able to make it earlier in the process	Since we know know the basics of how to do it, a future implementation will

	connect frontend with backend		be faster
Postgres	The group had some experience from the database course → Set up a database	Easier adding of new tables	Trial and error. Making the database easy to extend but hard to destroy
Facebook API	Did not know anything → Displaying data from specific posts	Fetch data from other social media APIs	Research on the internet

Gold plating:

First of all, we talked a lot about gold plating and how to deliver an MVP. Especially after the Minetest exercise. In the beginning, it was a bit hard to not gold plate since the PO was a bit unclear with both the highest priorities and also the end vision of the product. Thereby, it was easy to get carried away when making a prototype from an abstract idea. As the project went by we got better at delivering the product that the PO wanted by encouraging him to be more specific in his demands and wishes. The final MVP had no extra features but was creating the value the PO expected. In a future project, the gold plating issue would be considered earlier in the process by always keeping a close contact with the PO and that both parties continuously keep the MVP updated and relevant.

GUI:

Figma was used to create the prototype of the product, and we started off with some team members knowing more about the program than others. Thereby, we had a small introducing session to Figma so that everyone felt comfortable with building a prototype and making changes as the project evolved. All team members agreed that the actual code implementation of the final product would be easier the more specific the prototype was. Therefore, we spent quite a lot of time making the prototype functioning and the PO accepting it before we started the coding since changes in the code would be more time-consuming. Throughout the whole project, we however used the Figma model and modified it as the project evolved. In a future project, we will do advanced prototypes from the start since we already have the Figma skills needed to create a well-functioning prototype.

Frontend:

At the beginning of the project, the PO suggested that we use ReactJS for the front end. The majority of the team had none or little experience using it. During the Easter break, all the team members that were new to React watched a two-hour-long React crash course. It helped a lot that the whole team had some knowledge of the basics of React before we started the project for real. We also implemented some Bootstrap components in our React website. To be able to do it correctly we researched it on the internet and also did some trial and error before we got it to work. Apart from Bootstrap we also made some custom components that we styled with CSS. To use stylesheets were new to the group as well so we needed to research and learn together as the project went by.

In a future project, we want our front end to be even more advanced and the design to be even more consistent. We're going to accomplish that by implementing more Bootstrap components and using our already acquired React skills. If we start a new project with those skills we will be able to complete more tasks.

Backend:

When the PO suggested React for the front end he also suggested SpringBoot for the backend. The team didn't have any experience using it when the project started but with help from the PO and research on the internet, we were able to deliver a functioning MVP.

Postgres was used as the database. The group had some experience of this from a previous Database course. We made an initial sketch of the entity-relationship in Lucidchart before creating the database. In the beginning, there were some issues caused by an overly complicated design of the database. We had to remind ourselves of the MVP and to keep the database as minimal as possible.

We used Axios to connect the front end to the back end by implementing CRUD operations. None of the team members had used that before but as the project went by we learned enough to make it work.

In the future we want our backend to function even better without having to make as many mistakes on the way. We will do that by making the database as small as possible to start with, and then make it easier to extend. Since we already know how to make the connections between the frontend and backend, this connecting part will be faster.

Facebook API:

When we finished our MVP with a sprint left in the course the PO gave us one last task. To be able to fetch data from Social media APIs. This was new to all of us and we were short on time. We researched and managed to fetch some data from the Facebook API. If we would have had some more sprints we would have implemented more data from a couple of other social media platforms as well. In a future project, it would be beneficial to be able to implement more APIs, such as Instagram and Youtube. This would be made by doing more research on the internet and by watching tutorials.

Documentation usage and update

Topic	A (now)	B (future)	A → B (how)
Github's wiki function	Some of the group members had knowledge → Simple documentation of the APIs	Getting the right setup faster, to be able to start the actual project faster.	Learning by doing
Figma	Some knowledge within the group → A demo looking like how the final webpage is supposed to look.	Keep the Figma prototype more in line with the project throughout the whole process	Always having one person responsible for updating the prototype after each coding session
Lucidchart	No idea of how the structure of the database was supposed to look → Having a functioning database	Same as now	Just keep up the good work

Backend:

To keep track of the backend, we used Github's wiki function to provide simple documentation of the API:s we have used, as well as an entity-relationship (ER) diagram conducted on Lucidchart. All the team members had knowledge about ER diagrams from before but we struggled with how to construct the database to make it work as well as possible. Again, it is easier to keep the ER diagram up to date and change it when needed, before making changes in the actual code. It is also easier to read and understand how the database is structured by watching an ER diagram than to read the actual code. We ended up with a database with the right functionality that was not more complex than it had to be. The Wiki function in Github helped support the documentation of the backend and was regularly updated. It further made sure that every part of it was explained. In a future project, the documentation part could be more carefully thought through and be more consistent. We could definitely set a standard at the start that everyone knows how it works, and uses through the whole project.

Frontend:

We used Figma to document the progress of the front end. Every time we changed a feature in React we also updated the prototype in Figma. We used the prototype to show the PO our progress. When he had feedback we added comments to the prototype to keep track of things that needed to be fixed in the actual code. It was also convenient to keep the prototype as a template and goal for the front end since it's easier to visualize ideas in Figma than directly in the code. It also helped with ensuring that all team members had the same vision and worked towards the same goal. For a future project, a better standard for documentation within the frontend could definitely be established. Using something like javadoc and make sure everyone is comfortable with using it makes the code more easy to follow as well as introducing new programmers to work on the project.

Code quality

Topic	**A (now)**	**B (future)**	**A → B (how)**
Code review by team members	Half the group had used it before, the other half had not.	All group members are able to review each other's code and make pull requests	Went through it together the first times to make sure that everyone felt comfortable doing it on their own later
Codefactor	Did not know it existed	Scanning all code published on the repo to ensure code quality	A short research on what it was, and then implemented it

We installed an automatic code analyzer, Codefactor, in our Github repositories to check all our code. We did not install it to both our repositories at first, since we were not sure if it would be compatible with one of them, but after some research, we realized we could use it on both of them. Codefactor helped us get rid of things in the code that was never used, and gave hints when the code was vague in any way. This helped us reduce, or sometimes even eliminate, some of the bad practices we used along the way and made the overall quality of the end product better than what it would have been without Codefactor.

We also performed peer reviews on each other's code every time we made a pull request to the main branch. This ensured that we resolved merge conflicts and got a code of good quality. Some team members had done code reviews before, some hadn't. We went through it together the first few times to make sure that everyone felt comfortable doing it on their own later. This does not ensure 100% that the code is of high quality, but at least everything is checked by at least three of the group members, which made everything we published was of our highest level of producing high quality. We could have done a better and more consistent job on the documentation and code quality. It, unfortunately, became a low priority and we focused more on the MVP. If we would have done a similar project again, we would have documented it more carefully.

Application of Scrum

Topic	**A** (now)	**B** (future)	**A → B** (how)
Roles	We switched roles in the group each week and had some problems with authority.	Have clearly defined roles of responsibility.	Should define responsibility and authority for the roles early in the process.
Agile practices	We used a couple of useful practices which helped us.	Be more confident in using the different practices and a clear mutual agreement on how to use them.	Creating templates and studying more previous examples.
Sprint Review	Working with our PO became easier in the later half of the project. Reviews were divided in 2 parts, within the group first, then with the PO. We were not as strict about code structure as we would have liked.	Would like the PO to be more strict and decisive. Better quality/cleaner/consisten code. A DoD which we will also follow better.	Defining the relationship with the PO clearly and early. Define clear coding conventions and DoD for the project which are updated throughout the project.
New tools and technologies	Learning-by-doing, often by asking each other or online guides. Not understanding certain parts of the project caused problems a few times in implementing new features.	Have better understanding before trying to implement features. Knowing before starting new features what is necessary to learn.	RTFM. Try out new tools in a sandbox first to learn, then apply what's needed.
Literature and guest lectures	No guest lectures or assigned literature.	Establish mutual terminology through specific litterature.	Mandatory reading followed by discussion. Digest the theory bit-by-bit.

Roles

NOW

The roles we chose were: Scrum master, GUI Lead, Frontend Lead, Backend Lead. The roles helped us structure the project as a whole and aided in keeping track of progress on each sub-team's assignments during daily standups. The authorities of each role were quite vague which occasionally stopped us from keeping each other accountable.

FUTURE

Each role should have a clear area of authority to help us make decisive decisions and hold each other accountable for mistakes and/or praise. Having defined requirements for each role makes evaluation of performance much easier for the group.

HOW

Keep the role titles but give each role a stronger authority to make it more significant and distinct. For example, having the authority as scrummaster to callout good vs subpar performance can help set clear goal posts and bootstrap a strong team culture.

Agile Practices

NOW

- We had weekly spring meetings but felt like the time was never enough because we were hesitant and inexperienced with defining tasks.
- We had a lot of daily standups which kept us from sliding apart as a team.
- We started implementing KPIs only in the later half of the project due to misunderstanding their function.
- The Scrum Board was a good way to gamify the work and keep a solid overview but we had some issues with naming conventions as well as estimates for tasks/UserStories.
- We adopted the *slice-the-cake* methodology which improved the synchronization of the smaller teams, though there were times during the first weeks where we ended up with more of a waterfall structure which our supervisor steered us away from.

FUTURE

- Be confident in defining tasks and user stories as well as doing effort estimation to be able to have fast and decisive weekly sprint meetings.
- We would like to have well defined KPIs earlier in the project.
- To have a clear definition and mutual understanding of the elements in and structure of the scrum board to aid our workflow.

HOW

- Having clear examples of other scrum boards would help a lot early on with both defining tasks and making estimations. Practicing by making “mock”-sprints together early on would help a lot in coming to a mutual understanding of the scrum board’s role in our project.
- Defining mutual templates and examples in the social contract during the first weeks would help us a lot in accelerating the planning meetings and as well as increasing our confidence in breaking down the user stories into tasks.

Sprint Review

NOW

Our PO participated in the final stage of every weekly sprint review and expressed his opinions about our process/product. Prior to these meetings we conducted our own reviews, through pull requests and comments, to ensure that the code meets our standards and that it has fulfilled the acceptance criterias.

In the first half of the process the PO was a bit reluctant to apply pressure and proper critique but that changed during the later half after we expressed a need for clear goals and guidance surrounding his wants.

We had a DoD but it did not adequately represent what we actually did to check if something was finished.

FUTURE

Having the PO's wants clearly defined for the group to avoid gold plating and bolden the confidence of the team to build.

Strict rules regarding the coding guidelines to assure that the code is reliable and follows a consistent style.

We want to have a DoD that is both meaningful and useful.

HOW

Defining a clear developer→client relationship early on would help mitigate any misunderstandings in what the requirements of each party are.

We should hold a separate meeting in the beginning of the course to discuss the coding conventions for our project, define them together and progressively iterate as we learn more about our tools throughout the project.

We should follow the DoD more closely during the project as well as update it when it is necessary to do so.

New Tools and Technologies

NOW

As we were all new to the toolchain we had chosen a lot of the process was learning-by-doing either through asking someone else or searching for guides online.

Several times in the process members of the team got more-or-less stuck when implementing a new technology because the complexity in how it fits in our application was not fully known prior to plugging it in.

FUTURE

Learning a new tool through a side project where the learning is isolated and errors won't come from already built parts. A clear definition of what the learning is for in order to not stray away from the goal in the learning process can also be useful.

HOW

Here we would like to propose an ancient well-used technique in the world of programming and engineering: RTFM.

Trying new tools out in a sandbox environment with limited parameters at play should really help ease the frustration when trying to shoehorn it into the existing project. Six hours of debugging can save you 10 min of reading the manual.

Literature and guest lectures

NOW

There were no guest lectures. We mostly followed what we got from the ordinary lectures and what group members already knew since before.

FUTURE

Having a clearly defined set of sources and literature to establish a mutual terminology in the group around the agile framework can help a lot with internal communication.

HOW

Having some kind of mandatory reading for all team members which would be followed by discussion would be good.

Every sprint could be focused on implementing a specific aspect of the agile framework in order to more easily digest all the literature and get hands-on experience of abstract theory.