FINAL REPORT

Course:	Agile software project management (DIT257)
Group:	Fortnite
Members:	Joachim Ørfeldt Pedersen, Anton Hildingsson, Mattias Oom, Ashor Khizeran, Charles Sundström, Malin Jansson and Ludvig Östergaard
Github:	https://github.com/feldtsen/obtuse-dit257

1. Customer Value and Scope

Current situation (A)

When developing the Konrad application, we studied the UN Sustainable Development Goals and used them as inspiration for our user stories. Our intention was to provide value for any small community which might benefit from an easier way to share food within the group. A rough overview of the application was described in the first iteration of our epic.

This epic was thoroughly discussed within the group. Small changes were made, but throughout the course, the contents of the initial epic didn't change much. We agreed on the epic together, but although some aspects of the application changed during development, we never updated the epic as seen on github.

We worked on a sprint each week (starting week two), and towards week seven we had something that started to look like a finished application. It was user friendly, looked good, and contained most of the functionality we described in our user stories (see more at 4). At this point, we worked closely with our user stories and acceptance criteria, to ensure everything we worked on provided some kind of value to the (possible) users of our application.

A few sprints in we started doing proper estimations and prioritise the most important user stories (more on that under *application of SCRUM*), which helped us out in the end to create a more full-fledged application.

However, we realised we started out too big. We had plans for implementing online functionality and have more complex model logic. We didn't have time to implement these things, so we had to abandon a few user stories in favor of a fully functional local-only application.

Towards the end of the course, we did some fine tuning and added the last missing functionality and GUI elements, to ensure the application worked as expected and was easy to use for any new users. We also redesigned the entire application, to give it a more modern look, which is what is expected from today's users.

What might or should be (B)

A few things we should have done better is focused on the most important functionality. For example, early on we did not order user stories after customer value / importance. If we had done this we might have been able to create a more complete and valuable application while avoiding unnecessary work (for example, we ended up doing a lot of reworking, and we removed quite a few previously created classes).

What we also might do, especially if our goal is to create an application that would actually be launched and used in the real world, is to communicate and listen to the intended user-base. For example, we could have looked at similar, already existing applications. What features did they have? What do we think they lack? What does the community need? What should we do differently to get an "edge" and to provide as valuable a product as we possibly could?

As we worked through the project, we should have adapted our epics, to give us a more clear vision for the project. By iteratively adapting our goals and communicating with possible users, we might create a better product. Epics, user stories, acceptance criteria, none should be set in stone.

In a real-world scenario, it might also be possible to launch a second version of the application, with changes and improvements.

Feedback designed to reduce the gap (A → B)

Next time, we would conduct more team discussions and be even more thorough when doing our priorities. What functionality actually is the most important? Each sprint we could have a meeting where we discuss the planned functionality, where we are as a team, and what we'd like to do next, based on what we think would be the most important for bringing the application forward. The goal for an application like ours is to eventually launch it, and for this to be viable the application needs to be full fledged, that is, in a working state, but open for extensibility.

This could be done using rounds (allow every team member to air their thoughts, one at a time), open brainstorming, or another more structured meeting method. More about this is discussed in the section for the application of SCRUM.

In a real life scenario (that is, if we were to launch the application) we could reach out to potential beta-testers, or create some form of focus group with members of communities similar to those we imagine to be the end users. On a small scale, the most realistic solution is to reach out to people the team knows, but on a larger scale, we could possibly reach out to one of the UN sub organs, since we have several of the UN Sustainable Development Goals in mind when designing our application. If we get in contact with the targeted community, they could "play test" our application and provide feedback on how we could improve it.

And after launching the application, we could continuously listen to the communities using our software, perhaps by email or a form within the application, and create a second, better version which takes all the feedback we've received into account.

2. Social Contract and Effort

Current situation (A)

We wrote a social contract which we added to our GitHub repository. We discussed and agreed on the content of the social contract together. At first the social contract mostly consisted of our group values. We were supposed to be adding more specifics about meetings and communication, but we actually forgot to put this into the social contract itself.

As a team, we aspired to create an open and well-functioned environment, where every team member was able to communicate their thoughts and feelings. We agreed to be respectful, constructive and to value constructive feedback, where both the person giving feedback and the one receiving feedback will try their best to address any concerns that are raised. Overall we acted in line with the social contract, although we could have given each other more feedback.

In the beginning of the project we didn’t specify how many group meetings we would like to have each week. After the second week we mostly decided when to have the next meeting during the previous meeting, but not at all times. On the fourth week we added stand ups three times a week on Mondays, Wednesdays and Fridays at 12:30 PM to our social contract. We also added that every Friday we would have a meeting to make the weekly team reflections and on Mondays we would meet and plan the upcoming sprint. These specifics we unfortunately forgot to put into the social contract.

We were confused by the term *effort* and we didn’t really understand what we were supposed to measure or what results to aim for. Most of the time we were working quite efficiently. Especially the 6th week we got a lot done, even though everyone couldn't spend huge amounts of time on the course. A few people think they could have spent more time and energy on the course, even though we have gotten a lot done. However, we are happy with the product and what we have achieved overall.

We communicated using Slack, Discord and occasionally Microsoft Teams. At first we had a few channels in Slack and Discord but we didn’t really have a defined structure and the group was not always using the different channels cohesively. However, the communication improved during the project and we specifically stepped up our game after sprint 4. We reworked our structure and communication and had a long talk about how to keep working. (Since then, the sprints have been working quite well.) We created a channel in Slack for each user story included in a specific sprint. This enabled us to keep the user story discussions separated and also made it easier to communicate, and to get help and feedback from the other members working on the same user story. We also added a separate channel in Slack for communicating when and where to have meetings.

What might or should be (B)

Next time we should update our social contract continuously when agreeing on making additions or changes, and also discuss and include more details about the SCRUM workflow and communication between meetings. The social contract should be a fluid document, one which can be incrementally adapted and improved as the project progresses. Each week this document should be discussed and changed ought to be applied if the group deem necessary.

In future projects, we should aim for having a more pronounced “team work spirit” described in the social contract. The intention of this should be to inspire more pair programming and open communication. We should also have given the group more opportunities to communicate the kinds of values they would like the team to have.

From now on we should have a structured plan for each meeting and make sure to agree on the purpose of each group meeting in advance. We should also make sure to always agree on when to have the next group meeting, before ending the current meeting. Daily stand ups might be held more often to keep everyone updated. These things should also be described clearly in the social contract. If we write down how we should deal with meeting times and participation, this should be connected with the way we treat each other overall, and hopefully inspire greater meeting participation.

Next time we should have better ways of measuring the outcomes of our work, and evaluate if our effort was sufficient or excessive, and make changes if needed. This might help the group as a whole to develop new strategies and simultaneously understand each other better. In the beginning of the project, we should have described what our definition of "effort" actually is, and how it could be measured. We might also specify what a group member could do to increase their effort, if necessary.

In future projects we should have a structured way of communicating between meetings. We should also have more open discussions during meetings, more rounds, more listening to all members of the group, so that everyone knows about every part of the application. Put more effort into making sure everyone involved in a certain area knows when changes happen and keep each other updated.

Feedback designed to reduce the gap (A → B)

In general, we could develop our social contract by having "social meetings" where we specifically talk about collaboration, team-work and the social atmosphere of the group. Here we could discuss how well each member thinks the group works as a whole, how we treat each other, how everyone feels, and so on. Such a meeting could be held once a week or every other week, or whenever any group member feels it's necessary. During meetings like these, there might be one team member which has the role to mediate between other group members, to facilitate discussion and lead the meeting. This person should strive towards helping each member to air their thoughts, and if someone wishes to be anonymous, this person could be represented by the mediator. This would be especially valuable for personal and social issues, where the atmosphere easily could become hostile.

We would all make sure to ask for help and to help each other out when needed. To ensure the values of every group member is represented in the social contract, the group could do "rounds" where every group member can say what they want without interruption, similar to a daily standup. Similar meetings could be used for any group member to say if they think a group value is not respected. This could of course also be done anonymously, through a chosen mediator. By ensuring everyone has contributed and been part of the discussion surrounding the social contract, everyone would hopefully be more inclined to uphold its values.

By creating and using a separate Slack channel for communicating meeting specifics such as time and place from start, we would make sure no one is missing out or is confused about meetings. To prevent us from having unstructured or unnecessary meetings with no specific topics, we could set an agenda for upcoming meetings. We would also create the separate user story channels we used at the end of the project from start, and when a user story is finished we would archive the channel.

In the planning phase of the project we could do research about SCRUM and make sure everyone is understanding the definition of effort and then discuss how we could measure it.

Adapt our work depending on the amount of time and energy the team members have. Make sure no one is overworked, and that everyone has the tools they need to work at full capacity.

Regarding communication, we should have more physical meetings which would make it easier to do pair programming and communicate ideas freely. When physical meetings are not possible or viable, more slack/discord meetings should be held to ensure we can help each other out. With open communication, everyone will be able to work as efficiently as they can, to enable us to better work together as a team.

3. Design decisions and product structure

Current situation (A)

During this project, we've continuously been updating our UML-diagram to keep it in sync with changes to the source code. Mostly, we've been using a live document on Lucidchart, and only occasionally updated the UML document on github. One issue we had that we often made changes to the source code before changing the UML, and then we had to go back and change the UML to keep everything in sync.

We tried writing many tests early on, but in the first few sprints the code mostly consisted of skeleton model code which didn't contain any solid logic yet. It was difficult to test. Later on, as the model became more complex and contained actual testable logic, we were able to test many components (using unit tests) to ensure they functioned as we wanted them too.

The model quickly grew and we followed our initial design model quite closely. This, however, turned out to lead to some unnecessary work. We anticipated the need for certain classes, such as Tags, Items and TimeRanges, which in the end turned out to be excessive (Tags were turned into regular Strings, the functionality of an Item could be contained in a Post, and TimeRanges were removed completely). Instead of starting simple and working incrementally, we tried to anticipate future functionality. Although our goal was to design our code in a modular, object-oriented manner, we started off too large.

The model code turned out well in the end, but we had to change a lot. The same can be said for the GUI elements of the application, although we had another problem in this case. Many GUI classes were implemented on the go, and the interaction between them didn't necessarily (in the beginning) follow a determined pattern. For example, in the end we used the Singleton pattern quite a lot, with global instances accessible anywhere in the code. This made it difficult to follow the flow of the GUI-side of the application, and not obvious how to implement new functionality. The first few sprints, many group members didn't have proper insight into this section of the project either.

All this got better each sprint, and in the end all group members had, in some way, worked on the design of both GUI and model elements, however the path there wasn't completely straight.

However, certain services and view/controller components were difficult to test, and one reason for this was our heavy use of singletons. It was difficult to test using temporary testing files stored on disk, since many paths were hard-coded or dependent on a singleton instance of a specific class. It was also difficult to simulate user actions in tests, since our code needed actual button presses to run.

What might or should be (B)

In the future, we should have a clear software design ready before we start coding. The UML and design principles, and possible documentation should be discussed and ready before changes in the source code is made. By designing together and coming up with plans for changes, all members of the team would also get greater insight into all aspects of the project.

Similar discussions should have been held at the start, when designing the GUI side of the application. We should have had a plan for how GUI elements ought to communicate and how the GUI code should be structured. The domain model is important but the design of controllers and GUI elements should be documented as well.

Another way we could have improved the design is by doing more general documentation, both in terms of technical documents and code comments. By adding comments retroactively, as we often did, it was difficult to follow along with the work of other people, which made the cohesion within the group and the general design weaker.

When writing tests of our code, we should have been more strict when testing model code, as well as services which dealt with reading and writing to files. We ought to avoid overusing the singleton pattern and strive towards an even higher degree of modularity. If we didn't use singleton instances of certain objects, and if we abstracted the GUI and controller classes using interfaces, we could have tested many more parts of the application.

In future projects, we should also be better at respecting the model-view-controller separation (MVC pattern). For example, in our case, some login and registration logic ended up in the controller classes, code which instead should have been model code. This should be done differently in future projects, with large refactoring a possibility to solve problems of modularity. This is especially important for making the code more testable, since it's easier to test internal application logic if it can be run without the use of a controller.

In general, we should have started smaller, with a solid base which we could incrementally improve as the project progressed. With a modular design, it's easy to add new functionality when needed. Instead of trying to anticipate all functionality we might need in the future, we should have developed the core of the application and then considered expansion.

Feedback designed to reduce the gap (A → B)

When doing future projects, we could spend more time on the planning and designing part of the project. The domain model should be complete, and we should have created an UML diagram that is detailed enough for us to know how every class should interact. The UML diagram should not just be a dependency graph, but it should detail how complex model interactions should occur. We could also create other kinds of diagrams, to describe the intended flow of the application — how should model, view, and controller interact? how should we structure our code to accomplish this behavior? and so on.

And when we work on the product, we should consult the plan when unsure how to implement something, not implement on the go and then change the plan. This might be a slightly naive plan, since it's probably impossible to design something perfect on the first try. Therefore we need to keep the plan open for change, just like our user stories and epics are open for change, however, changes to the plan should only be done after discussions within the team.

One way to ensure code quality and that we follow the design principles we've set out to follow, is to work with pull requests more closely. Each time a non-trivial change is committed, this should be turned into a pull request for someone else to review. If we review each change closely (i.e, not just glance over it and then approve) we can make sure the risk of breaking our plan or our principles is reduced.

For designing the plan, in the future we would study GUI design and javaFX conventions more closely. We ought to study how GUI apps are usually designed in javaFX, read about the accepted conventions, follow tutorials if necessary. We're not experts in our field. By studying properly before we start coding, we ensure we'll create the best application we can make.

Diagrams, UML, flowcharts, etc, should be made for GUI and controller elements as well, to get a proper overview not just over the model code. This would make it more simple to get the right application flow, and to structure model-view-controller interaction. If we simultaneously write proper documentation, then we and other developers could get a better understanding on how the application actually works.

One concrete design decision we could make for a future project similar to this one is to avoid overusing the singleton pattern, which in our case made the flow of the application difficult to follow. Instead, we should, as previously mentioned, study accepted standard patterns for applications like ours.

4. Application of SCRUM

Current situation (A)

The first three weeks we were in some ways working with SCRUM in mind but didn't use many of the specific tools apart from SCRUM board and the initial planning phase. We definitely made progress during the project and from the fourth week the application of SCRUM was substantially increased.

In the beginning of the project we all shared roles. From the fourth week Malin was decided to be assigned as the SCRUM master, but we didn't define any other roles. Later during the project when we got more used to using SCRUM, the SCRUM master role got more fluid.

The first sprint we set up a SCRUM board but we didn't have clear instructions on how and when to use it when started. The second sprint we moved our SCRUM board from GitHub, where we started, to Trello. The SCRUM board in Trello gave us a good overview of the project and was a great tool overall, but we sometimes struggled to create new tasks ourselves and mark the tasks correctly during the process. This led to some insecurities and confusions in the group, causing some duplications and bottlenecks in our workflow. Our acceptance criterias and tasks were sometimes quite big or unspecific. We improved during the project, but it was at times hard to start working on new tasks and a few times led to people working simultaneously on the same thing, by mistake.

From the fourth week we began having "daily standups" on Mondays, Wednesdays and Fridays at 12:30 using Teams the first time then Discord. If we would have planned a group meeting on one of these days we would incorporate the daily standup in the group meeting instead. The stand ups were a great help to communicate our work within the group and we continued with them throughout the rest of the project.

Starting sprint one, we agreed on three KPIs: Fist of five, team satisfaction and percentage of finished user stories planned for each sprint. After the second sprint (week three) we realised it was more logical and beneficial for us to measure the achieved acceptance criterias per sprint instead of whole user stories. The fifth week we had low KPI's and this we thought was due to lots of work in other courses combined with few physical meetings.

We started using estimates the fourth week to try to estimate the difficulty of each user story and to estimate our time to spend on each sprint. This went OK, and there was a great improvement that week compared to the week before. However, we often struggled finishing at least one of the chosen acceptance criterias for the sprints.

Generally, some group members found it a bit difficult to study one course where we worked using the agile framework, and another, "normal" course. By constantly switching between two ways of working, getting into the right flow was difficult. It would have been easier to *only* work within the agile framework, to get the most out of SCRUM.

What might or should be (B)

When doing future projects we should definitely learn more about SCRUM and make sure to get the process right from the start. If we properly understand how the agile workflow functions we could have greatly improved our weekly results, both in effort and quality. SCRUM consists of many parts, and although a group does not need to use them all, doing just a few would consist of a fractured workflow. We should have done estimates earlier, used our KPI's to lead our work, had standups daily, and so on.

The same can be said for SCRUM roles: if we did a similar project again we would make sure to define a solid and clear structure within the group, with defined SCRUM roles throughout the project. We were too tentative to implement the roles, and this could be since we didn't know each other's personality, skills or background. In hindsight not to have defined roles from the beginning was not the most effective or clear way to do the project.

A proper SCRUM board, like the Trello board, ought to be used from the start, to get a cohesive use of the SCRUM board throughout the project. With a more established SCRUM board, we could spend more time defining concrete user stories with clear acceptance criteria. The associated tasks should be equally well-defined and simple to tackle. There should be no ambiguity in an acceptance criteria, and in the same way, tasks should be more detailed and concrete. Anyone, with the required background-knowledge, should be able to pick up a task and implement it fairly quickly.

During our project, we decided to have daily standups not daily since the course was not full time but 50%, and we thought having it every day might be too cramped for the project. Next time we would have standups more often, to prevent the work stalling between standups and to keep the group updated.

The KPIs should be taken more seriously overall. Next time we should let our KPIs lead our work, make sure to use the KPIs to evaluate our work continuously, in order to help us acknowledge issues and progress. In addition to this, some of our KPIs weren't perfectly helpful, and we ought to have realised this sooner and changed them. Or, we might change the way we worked with our KPIs. The "fist of five" KPI wasn't much helpful since it never inspired concrete change or discussions. We could have had KPIs that benefited our work to a higher degree. In the future, we ought to more closely consider what we should change if a KPI reveals a poor result, and we should carefully pick KPIs that aid our specific project.

Another important thing we should have done better is to improve our estimates. However, overall we have learned that estimating correctly is quite challenging and difficult. Estimation is therefore a process which gets better for each sprint, and for each project. It's difficult to assess how quickly the team works before the work actually gets going.

Feedback designed to reduce the gap (A → B)

Before structuring the first sprint, we could take some time, maybe a full week, to study the SCRUM workflow. Loads of resources exist online: tutorials, documents, or even detailed "cheat sheets" which we could study. During this week, we ought to consider which aspects of SCRUM best suits our project, and create a plan, document, or "cheat sheet" of our own, with the basis in the studied documents. We might also want to read about common pitfalls and things to avoid, to ensure we do not make the most common mistakes.

We would use SCRUM roles from the start and have stricter definitions of the purpose and responsibilities of the roles. For example a SCRUM master who keeps track of the user stories and tasks and ensures everyone in the group always knows what needs to be done, and has the knowledge and specified task instructions required to do so. This would ensure no group members are unsure on what to do, or wants to do something but lacks the proper communication channels or knowledge to do so.

When starting the project, we could set a standard for where and how to use the SCRUM, and add this to the social contract. Before starting a task, one should always consult the SCRUM board, and if necessary talk to other team members which are involved within the same area or user story. This avoids unnecessary duplicate work.

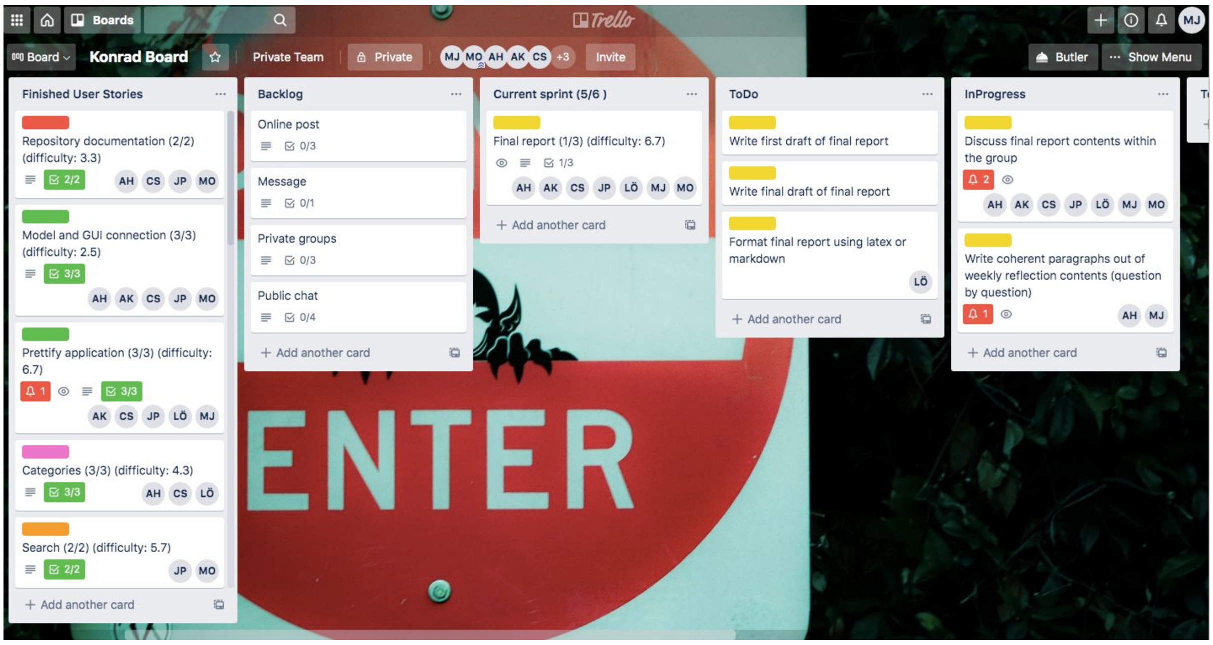
Each acceptance criteria could represent a concrete and perceptible aspect of the user story. They should not just be made up, used by us to arbitrarily decide when a user story is finished or not. An example of such acceptance criteria could be aspects of concrete functionality, for example “Is there a login button which takes the user to the login page?” or “Is the posts sorted from oldest to newest?” During our project we often had tasks similar to “Implement tag dropdown”, without any instructions on where, how, or why to do this. Tasks would not be vague, they would detail exactly what should be done and give concrete suggestions for how to do so.

Regarding daily standups, we would actually hold the daily and not just every other day. We would start doing them as soon as we started the project and this would make sure everyone knows what to do next, and how to do it. The SCRUM master could have a more active role in pairing people together, if someone is unsure or needs assistance.

We could explicitly decide on priorities and difficulty estimates for each user story during each sprint, not just the later ones. The group could have an overall time-plan, that is, a schedule for when certain functionality should be implemented, when the first iteration of the application should be sent to beta-testers, when it should be launched, and so on. A schedule would help the group focus and prioritise. When a user story is in progress, estimates could be evaluated after each daily standup. If we each day check the progress of a user story, it will be easier to see if a user story will not be finished in time, if additional help is required, or if we need to change some tasks or acceptance criteria.

If chosen wisely, KPIs are a great way of keeping track of coding progress and things like team satisfaction. Once a week, we could discuss the KPIs and possible changes or improvements that need to be made. We would be more open to change and ready to adapt the process if necessary. We would also choose and use KPIs that better suits our project. For example, instead of having two KPI’s that measured how the group felt in different ways, we could have had another KPI measuring code quality (evaluating using for example Travis), GitHub commits, user satisfaction, or something else that might impact the project.

Appendix



Boards

Trello

Board Konrad Board Private Team Private MJ MO AH AK CS +3 Invite

Butler ··· Show Menu

Finished User Stories

- Repository documentation (2/2) (difficulty: 3.3) — 2/2 — AH CS JP MO
- Model and GUI connection (3/3) (difficulty: 2.5) — 3/3 — AH AK CS JP MO
- Prettify application (3/3) (difficulty: 6.7) — 1 — 3/3 — AK CS JP LÖ MJ
- Categories (3/3) (difficulty: 4.3) — 3/3 — AH CS LÖ
- Search (2/2) (difficulty: 5.7) — 2/2 — JP MO
- + Add another card

Backlog

- Online post — 0/3
- Message — 0/1
- Private groups — 0/3
- Public chat — 0/4
- + Add another card

Current sprint (5/6)

- Final report (1/3) (difficulty: 6.7) — 1/3 — AH AK CS JP LÖ MJ MO
- + Add another card

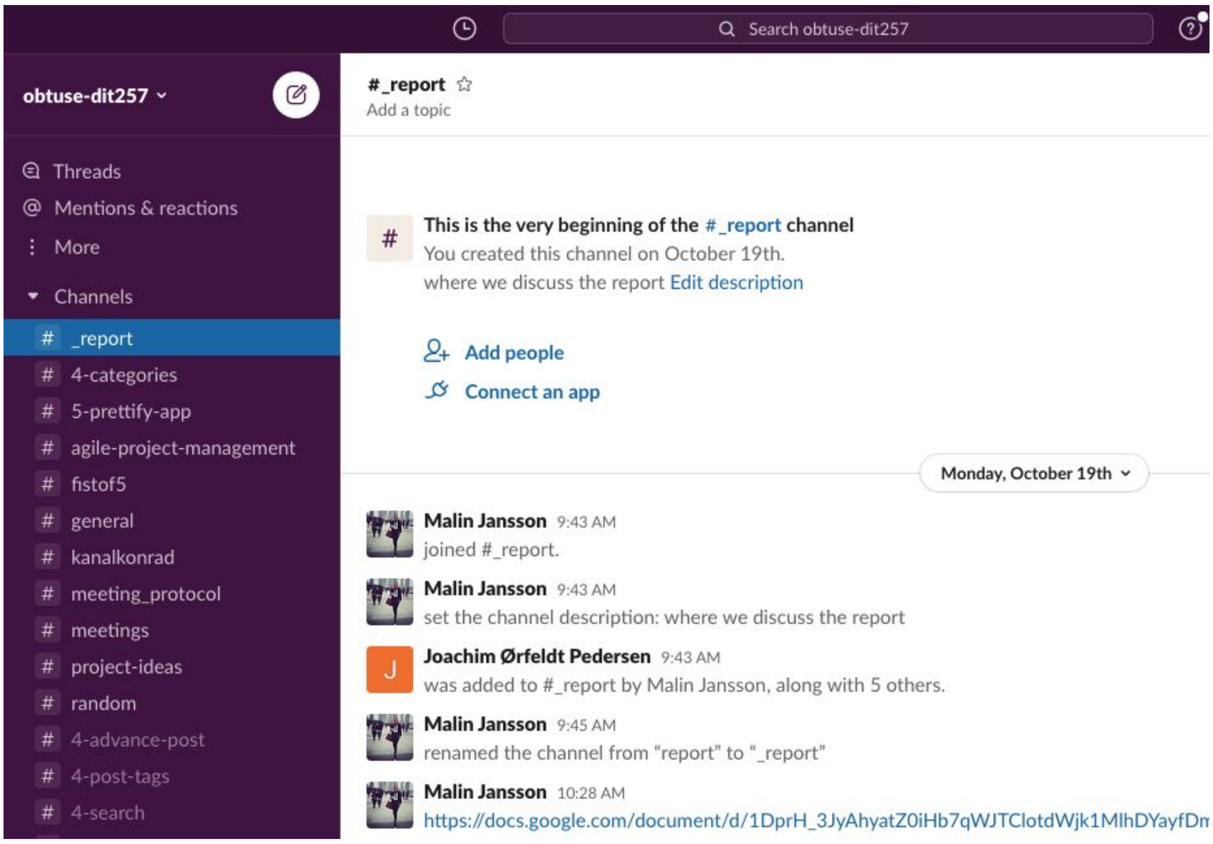
ToDo

- Write first draft of final report
- Write final draft of final report
- Format final report using latex or markdown — LÖ
- + Add another card

InProgress

- Discuss final report contents within the group — 2 — AH AK CS JP LÖ MJ MO
- Write coherent paragraphs out of weekly reflection contents (question by question) — 1 — AH MJ
- + Add another card

Picture 1. Our Trello board.



Search obtuse-dit257

obtuse-dit257

Threads
Mentions & reactions
More

Channels

- # _report
- # 4-categories
- # 5-prettify-app
- # agile-project-management
- # fistof5
- # general
- # kanalkonrad
- # meeting_protocol
- # meetings
- # project-ideas
- # random
- # 4-advance-post
- # 4-post-tags
- # 4-search

#_report ☆
Add a topic

This is the very beginning of the #_report channel
You created this channel on October 19th.
where we discuss the report Edit description

Add people
Connect an app

Monday, October 19th

Malin Jansson 9:43 AM
joined #_report.

Malin Jansson 9:43 AM
set the channel description: where we discuss the report

Joachim Ørfeldt Pedersen 9:43 AM
was added to #_report by Malin Jansson, along with 5 others.

Malin Jansson 9:45 AM
renamed the channel from “report” to “_report”

Malin Jansson 10:28 AM
https://docs.google.com/document/d/1DprH_3JyAhyatZ0iHb7qWJTClotdWjk1MlhDYayfDn

Picture 2. Our Slack channels.