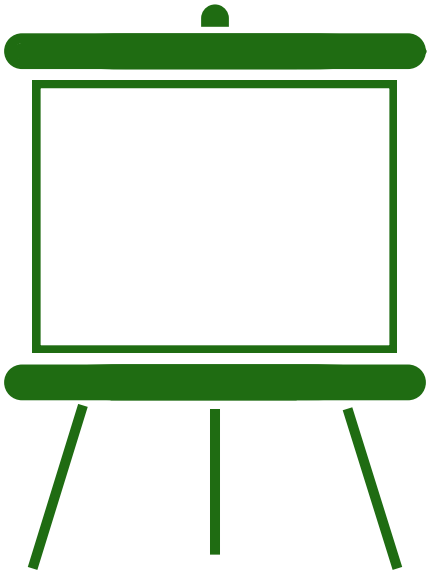


GALLERY

A collection of Canvases

Group F Project Report

Adrian Lind Bergersen, Tyra Fosheim Eide, Martin Eide,
Thorbjørn Melheim, Audun Oklevik, Philip Svenningsen

Contents

Introduction

Gallery is a web platform that aims to combine multiple instances of the learning management system (LMS) Canvas [6]. Canvas is used by, among others, the University of Bergen (UiB) and The Western Northern University of Applied Sciences (HvL) to share important information and course material with their students. The login to Canvas is done by providing each student with a Feide [21] account. Feide is a single sign on service, and the students use this account to log in to Canvas and other institution web pages. An issue arises when a student is enrolled with multiple institutions using this solution - they receive one Feide account for each Canvas instance, and these accounts are frequently confused in the browser when trying to log in, resulting in refused access and increased time spent on logging in for the student. UiB and HvL have a joint master's degree in Software Engineering, and this is an issue that affects all of these students and others each semester.

The goal of this platform is to eliminate the time students spend on logging in to Canvas. Our platform should provide all the information the institution Canvas provides as well as improve upon Canvas' features. The students should be able to:

- Read announcements posted by the lecturer
- See all the courses they are enrolled in
- Deliver assignments
- Get weekly update mails to a specified email
- See their Calendar updated with events from both institutions

The goal was to include features directed at lecturers, like posting announcements and have it automatically be cross-posted to Discord, but for the project duration we were only able to prioritize the student perspective of the platform.

Motivation

The problem at hand was chosen because all team members are enrolled in both UiB and HvL. We have all encountered the issues described in the introduction, and we all know other students who face the same situation. Specifically, we all had problems getting to the assignment information for this project at HvL's Canvas, as we had issues switching from our UiB account to the correct HvL account and staying logged in.

We also performed a survey to investigate how relevant our identified problem was for other students. We received 28 responses, and an overwhelming majority reported having a negative experience using two LMS platforms - 78.6% of the respondents chose 1 or 2 on a scale from 1 to 5, with 1 being "extremely negative" and 5 being "extremely positive". No student responded more than 3 on the scale.[1] Clearly, other students face the same challenges that we do.

UiB has a page, UiB Treet [23], that aims at gathering important information for students in one place. This is not connected to Canvas, however, but encapsulated the concept we were going for. We originally wanted to create a solution where you could log in to Gallery and then automatically be logged in to all Canvas instances, which was inspired by the Helsenorge app [17]. This app lets the user log in on their phone with biometrics, and then redirects the user to the Helsenorge website without requiring to log in with bank information like when going straight to the website. This idea was discarded, however, as it turned out to be extremely difficult to solve.

A page collecting all information needed by a student in their every day life does not exist, and with the current log in situation, this is needed.

Implementing such a platform would not only save students time, but it would facilitate a better overview of their course requirements. Reading announcements and doing assignments are essential parts of any course, and missing out on this could be detrimental for a student. The platform would not only benefit students in the joint master's program, but also students taking singular courses while enrolled in another institution.

Our full development process is described in the further sections of this report.

[1] The survey is not included in the report due to formatting difficulties. For access to the whole survey, please contact one of the authors.

Pre-study

Design Thinking

When it comes to our usage of design thinking activities, it had an overall good impact on the project outcome. Our process started with the team brainstorming different problems in our lives that we wanted a better solution to. After coming up with some of the problems the group faces in their day-to-day life, we tried to implement the double diamond model of design thinking [18] in order to find a solution our problem.

We implemented both qualitative and quantitative interviews in our *emphasise* phase, which helped us understand that there is a real need for a better solution to the current Canvas environment (when the user belongs to several institutions). This reaffirmed our belief that the problem we had, was also a problem that many other students had. We then created personas and tried to pinpoint the existing challenges for our user group. The personas was part of our *define* phase, and at first we tried to make personas based on our hypothetical average user, which was based our knowledge from the interviews in the *emphasise* phase. After that, we created some personas that represented other potential users we had not originally considered, and it gave us a deeper understanding of the needs and wants of several user types of our platform. All personas can be found in appendix A.

Next up, in the *ideate* phase, we had a session where the group tried to come up with some wild solutions and ideas in pairs and then talk about them with the entire group. This was slightly helpful, but it didn't really expand our understanding of what we wanted to do, as the proposed wild ideas and solution were way too crazy to be actually implemented. It did, however, promote both creativity and it also helped the group in getting to know each other better.

Lastly, both the *prototype* and *test* phases were not really used. We did create a rough mock up of how we wanted our platform to look, but we didn't get any feedback on the mock up itself. We also didn't re-interview anyone to get there feedback on the mock up. The mock ups can be found in appendix B.

Overall, our usage of the design thinking process, could relate to the "Ideas over implementation" paragraph from "Design thinking was supposed to fix the world. Where did it go wrong?"[1]. The design thinking helped us in getting a deeper knowledge of what can potentially be done, but it did hinder us in getting where we wanted with our product in the time-frame

we had, because the design thinking process took a considerable big amount of time that could have been spent creating the actual product.

In conclusion, the design thinking framework was most useful to us at the start. The *emphasise* and *define* phases clarified our problem and pointed us towards a clear end goal. However, as the project progressed, there was a clear downward trajectory when it came to both the need and use of the design thinking process, mainly as a result of time constraints.

Sustainability

When we applied the Sustainability Assessment Framework Toolkit [13] (SAF Toolkit) to our project, we based the application on a workshop presented by Markus Funke. We followed the order of events in the Design Cycle, where we started with creating a Desision Map (DM), and then created a Sustainability-Quality (SQ) Model based on the Quality Attributes in the DM. We chose to also include the creation of a KPI-model, even though this is part of the Monitor Cycle, because this was done at the workshop with Markus Funke. We formulated our Sustainability Goals based on the DM and SQ Model, and created a KPI Model based on these goals.

We did not intentionally follow any of the strategies in the article [13], like creating a Decision Graph to determine the Sustainability-Time Dimension and an Interdimentional Dependency Matrix Template to determine effects between Quality Attributes. Other than this, we followed the structure of the SAF-Toolkit as described in the article quite closely.

In retrospect, it might have been beneficial to follow the strategies in the paper, but we feel that our brainstorming process made it easy for us to get started without using these strategies.

The most challenging part of the process for us was creating the KPI Model. We also faced some time management issues while creating this model, and the purpose of the model might have been lost in the fact that it was a required element for the first project evaluation. As a small group with a short time to work on this project, it was quite difficult to identify the critical success factors, as these are supposed to be tied to the organization's high-level goals - which we did not really have. We also had some struggles when creating the SQ-Model with defining metrics for some of the quality attributes, but we managed to do so eventually.

Our sustainability goals were as follows:

- min(login time)

- max(overview)
- max(integrity)
- min(cost for users)

These goals were defined in the planning phase of the project. We never actively brought up the sustainability goals throughout our project, but these goals were so integral to the platform we were developing, that they naturally occurred in our discussions. Our goal of overview and integrity were defined as full access to all Canvas elements and that these elements were updated and correct. We did make sure that the features we developed were relevant for accessing Canvas elements, and we has several discussions of how to keep this accessed information as up to date as possible. The cost for users goal was quite irrelevant in our project, considering we all worked for free and had allocated time through the course to work on the project. If we were a company or perhaps did this project in our free time, we would have had to consider how to keep the project running financially, and this goal would have been much more relevant. We were very occupied with minimizing login time in the start, but our plan for how to do this ultimately failed. Our other option was to reach our goals of overview and integrity and that way eliminate the need the user had to log in anywhere else.

Our understanding of sustainability evolved the most in the creation of the sustainability goals. It might have been beneficial to more actively engage with them through the production, but time restrictions and team size made it difficult to do so. It would perhaps have been nice to also have some goals of a environmental dimension, and this might have further developed our understanding of software sustainability. However, the need to produce something on time took over for the critical reflection around sustainability, and it was all sort of forgotten. This is something that we would want to do differently if we were to do the project over again.

The SAF Toolkit was, however, very useful to get us all on the same page about our visions for the platform we were developing. You can read our full process with the tools in the appendix C.

Modeling

Our use of modeling in the project mainly consists of descriptive models. As in models for the sake of documentation or describing existing concepts or code, rather than models meant as a blueprint for implementation; prescriptive models[4]. To understand why, we need to grasp

the importance of intent when building models[2]. Our project was not approached with model driven development (MDD) in mind, rather our models were constructed with product criteria in mind. Since the models were constructed to serve these requirements rather than as part of our development process, they did not serve to guide our thinking or approach to development and implementation. They were also thus primarily viewed as artifacts.

We can see this in some of our models, i.e. our domain model E.4 which simply did serve to document our understanding of the domain as is. It thus had little utility for our interpretation of our user and stakeholder interviews or questionnaires. For this same reason it did not widen our understanding of the domain much, besides giving us an opportunity to critically reflect on the domain.

We do see the utility in models as prescriptive tools. We believe that MDD could be useful for creating an application with sound architecture within a reasonable time frame. Though model creation could pose a significant overhead, with the use of low code tools it could serve to lower development time, which was a significant constraint for our project. Though we would not fully trust the integrity of code from tools such as AI code generators, and would seek to review and possibly refactor it. Also to avoid filling our codebase with unfamiliar code.

Our activity diagrams, illustrating how different feature workedE could've served as critical tools for constructing the architecture of our components. Serving to define the requirements of our code, with a similar benefit as TDD. Aiming to define the architecture and general requirements rather than the actual case-by-case runtime output. Though it could be said that it would introduce a similar overhead as TDD, it might also save time by avoiding confusion around the architecture and requirements of our code. Working with models as tools for abstraction engineering could also serve to improve overall code quality and architecture. As explicitly isolating the abstraction engineering process in software development could serve to improve quality outcomes and avoid uncertainty during implementation.[10]

From our experience, creating abstractions such as the domain model E.4 posed issues as the domain carried a lot of complexity which was hard to abstract. Thus it was difficult to draw the boundaries of what to include and not. This can be tackled in different ways depending on the intent and consumer[2]. Though we did not yet see the importance of this, and the domain model thus had little intention in it's design. On the other hand, when making the activity diagram illustrating our process E.2, we had a firm grasp on the process and it was easy to distill it to its most important elements in a diagram.

Architecture and Requirements

This project has largely been using and basing the Scrum sprints on user stories that we defined in the first meeting each sprint. They way that we created user stories were mostly based on the group brainstorming the most important (and sometimes less important) parts of the application. After coming to a consensus on the user stories that we wanted to put in the backlog, we decided on the allocation of story points together as well. At the start of the project we spent a lot of time creating user stories, and especially a lot of time on allocating the story points (sometimes using planning poker). Later on we decided to drop the usage of planning poker, and instead just talk about the story points in a less 'strict' setting. Furthermore, as we spent less time on the story points themselves, we instead focused more on creating more specific tasks for our user stories.

When it comes to the usage of user stories in this project, there has been both positives and negatives. The biggest positive was the fact that the entire group got a joint understanding of the problem habitat, which helped in several aspects, such as making communication and planning easier. The group could easily decide on who should work on what part of the application, in other words it gave us a more organized way of working and helped in creating an environment that promoted separation of concerns in addition to making it easier to work together with those that are working on similar tasks. The main negative side of using user stories was that it was time-consuming, and hindering us from doing more tangible work that the stakeholders could see.

When it comes to the amount of work, specifically the up-front work that we did in this project, there was certainly areas where we applied too little up-front work. One clear example was just the fact that our choice of programming languages and frameworks could have been better, as we overestimated the amount of knowledge we had of the chosen framework (for example Spring Boot), and didn't really second guess that decision until it was too late. This led us to spending more time than necessary on making things work, than it would if we just spent a little more time up-front in choosing a framework or language that would have been easier (such as Django in Python). In other words, we should have spent more time addressing the risk of our chosen technology stack [24]. Luckily for us, the team was well suited in responding to change and developing in new environments, which meant that our non-optimal architecture decisions didn't end in catastrophe, but it certainly made us spend more time than would've

been necessary if we spent more time on doing the up-front work [24].

The software architecture of our project is heavily focused on separation of concerns. The codebase is mainly split into a frontend and backend that works together through a REST API. The backend is then closely knitted together with the database, and they communicate together with the Java Persistence API. Another essential part of the software architecture is the DevOps, which focuses on using CI/CD through GitLab to build and test both the frontend and backend. This is a fully working interconnected environment, however, using this architecture is also somewhat of an antipattern for our group [11]. This is because the architecture has been dictated by what we have looked at as the industry standard, which should work perfectly in theory. But, we should have looked more at our own strengths and weaknesses, and catered more towards what we have, and not what the rest of the industry has. This led us to making more short-minded decisions, increasing the technical debt, and hindering ourselves from creating a better product, as it was difficult to know how much we are capable of doing when using languages/frameworks that we aren't too familiar with or are unnecessarily complicated [11].

CI Pipeline

In our project, we used a single Git repository for all code and GitLab CI/CD functionality to create a CI pipeline. This allowed us to automatically run the suite of tests on every commit, on every branch of the Git repository hosted on UiB's GitLab platform. This allowed for auto-testing before merging. Our branching strategy is a simpler version of GitFlow, with a protected main branch and feature branches. Our merging strategy is through merge requests where the CI pipeline results are a part of the code review.

This frequency of testing and distributed version control will not be able to scale to very large software systems as mentioned in [19] [15], but our smaller-scale project has sufficient computing resources.

Making a reliable and trusted CI pipeline required considerable effort. The problems we faced were hard coded URLs that worked on local machines but not in pipeline, tests requiring multiple dependencies to be running and accessible, long downloading of packages and long execution time of the pipeline as a whole. All of these issues were addressed by dynamically targeting URLs, using built-in capabilities of the GitLab CI platform and adding upgraded VMs to run the CI pipeline. We also had to consider whether we wanted easy tests to write and understand compared to how easy the tests were to run in the pipeline. We decided on prioritizing the ease of writing and understanding test over the CI pipeline, which complicated the CI pipeline setup. But we did not want the already cumbersome process of writing test to be more complicated to be easier to run in CI.

At first the benefit of an CI pipeline was not noticed. We expected that the more code and test we wrote, the benefit would be more apparent, which was the case. We experienced cases where the automated tests highlighted regressions in functionality that had a non-obvious dependency on new functionality.

One unexpected aspect of setting up the CI pipeline was the impact of false positives, which degraded trust when the pipeline failed. Initially, some failures were due to incorrect integration with our testing frameworks. However, there was a case where what we assumed was a false positive turned out to be a true failure, caused by the repeated failure of an unfinished part of the CI pipeline.

Software Processes

Throughout development we tried various kinds of development methods. We initially based our approach on the Scrum and XP methods and we tried different methods iteratively. In practice we followed a hybrid development method, were we combined practices from different standard policies like Scrum and XP. We mixed various standard policies to adjust to our needs, and we had a issue board with a sprint backlog that facilitated issues for the current sprint. In the paper of Klunder [9] it is described that developers in general use hybrid development methods. Our development strategy is more aligned with this reality than if we had followed a standard development method strictly. We had sprint meetings were we planned sprints, and discussed which issues to prioritize together. There was a mix of both test-driven development and agile development, and we added tasks continuously. We met three times a week, At the meetings we did not have defined roles, but we had common ownership for the code and took turns turns leading the project, We also had a practice of code review of merge-requests by another team-member, which we agreed upon in advance. Throughout the project code were reviewed to filter out overlooked errors and bad design choices early on. You can read more in depth about our software development process and the practices of our team in Appendix F.

Pair Programming

We did some pair programming but found that it required more effort than it was worth. There was not a noticeable increase in quality, The programming technique suffered a bit from the fact that there isn't that much experience with all the technologies we use, therefore it often ended up being "pair research" instead,

Pair programming was sometimes useful to overcome hard obstacles. The article of Dybå [3] claims that pair programming is helping the quality of the code in some cases, depending a bit on the skill level of the programmer and the complexity of the task at hand. The results showed that inexperienced developers would produce better code in general with pair programming, and experienced programmers did not. The results also showed that the positive effect of pair programming was best when a programmer is encountering a task that is challenging for them and their level of competence. We do believe this principle to be right, even though pair-programming did not benefit us too much.

One reason that pair programming was not perceived more positively is driven by the fact

that we as students work more asynchronously due to having different schedules, illness, exams, and having to prioritize other work. When we did work together in the meetings the pair programming dedicated a lot of time to explaining to the other person what we have done so far, before starting work on further work. If we found a task challenging it was more helpful to be two people trying to solve the problem simultaneously from different computers.

As we have gotten more experience with the different technologies, pair programming has become noticeable more effective. Switching roles between Driver and Navigator seems to help manage fatigue and keeps both people engaged. Overall, with some experience with the technologies, we see that pair programming has the potential to speed up development, learning and problem solving.

Refactoring

There was not a lot of code that needed refactoring in our project. However, we saw a great improvement in the parts that were refactored, even though it did not make a major difference to the project as a whole. This might have changed if we had worked on the project longer. Our refactoring can be especially valuable in the future if we decide to add a lot of code and end up with a more complex software architecture. From the findings of Meyer [16] we see that refactoring increases productivity, maintainability and quality. We do believe this to be true, because refactoring can clear out unnecessary complexity and thus provide great benefits over time. Therefore, we do believe that refactoring is very meaningful, even if it did not feel like it made the biggest difference in our case. Appendix D details some specific examples of refactoring we did throughout the project.

Task Specification and Test-Driven Development

Our project benefited on having a fine grained and detailed task description. This increased the quality of the project, as we could easily understand what was expected. The findings from the Karac [8] suggests this when test-driven development (TDD) is used. In TDD the ability to break a task into smaller tasks iteratively is very important. Therefore, a hypothesis was that granular task descriptions was especially useful for developers that use TDD. From our experience the granular task descriptions were helpful, not only when we used TDD. Detailed and granular task descriptions make it more easy to identify smaller sub-tasks that otherwise can be easily overlooked, no matter which development method is used. You can read more about

our experience with Test-Driven development in the section Customer Tests and Test-Driven Development (TDD) of the appendix.

Challenges

Throughout the project we faced many different challenges. The paper of Meckenstock [14] shows us various challenges of agile software development and groups the different challenges into different issue themes, The issue themes are process issues, team and developer issues, customer issues, technical issues and organizational issues, and all of these issues can possibly lead to project outcome issues, We will now reflect on some of the different issue themes in light of our project.

Process issues mentioned in the paper include planning, estimation and efficiency issues. It was a big challenge to estimate tasks, and this resulted in a big meeting overhead. See the section Effort estimation for more details.

Technical issues mentioned in the paper include incorrect application of ASD practices, like lack of documentation, refactoring, requirements and testing. In our experience we sometimes chose between functionality and documentation. Since we had many issues to do, it is a varying level of documentation present in our project, and some places there are no documentation. Refactoring has been done were we saw the need. We have been discussing precisely what our definition of done is, to align expectations. We included testing and correct functionality as a requirement to say that it is done. Setting up frontend-testing, end-to-end-testing, and the CI/CD pipeline turned out to be difficult and time-consuming. Adding more tests after that did not take a long time and was very useful for the project. Our testing efforts turned out to be a meaningful investment. The requirements in the reports and project assignments were detailed and precise in general, which helped us in the development. If there was any ambiguity, we could discuss it at the project meetings that we had weekly. The requirements and expectations of the reports and project were in general clear.

Some minor team and developer issues were that some of us encountered were a lack of competence and creativity. We sometimes wrestled with challenging tasks for a while before we found the solutions, However, these challenges were very small because we had good cooperation in the team.

Effort estimation

It was very time consuming and challenging to measure tasks in story points. We used planning poker for estimation of tasks at one point in the project. We soon decided to quit using planning poker to increase efficiency. Many of the tasks we started doing were more time consuming than we initially thought.

For especially big tasks that were not essential we made a special plan. The plan was that we set an amount of time that we were willing to spend before we decided to quit. This way we did not spend too much time. We could decide later on if we wanted to continue to prioritize the not-essential big tasks. It was a good opportunity to become increasingly better at estimating how long time a task actually takes. We also saw that experience is useful for completing tasks efficiently, and estimating the time needed to solve them. Sometimes it would take additional time to understand how to solve the tasks. At the end of one sprint we saw that our burn-down chart did not reflect as many story points that we planned to do by far. The way we dealt with this problem was to add more value and story points to the tasks we were working with, to reflect the efforts needed to complete the tasks more accurately.

Time pressure

Time pressure is an especially important challenge that we have had at times in this course. We prioritized to meet deliver before the deadlines of the reports and prepare for the presentations of our application. One finding of the paper of Meckenstock is that time pressure can be a problem chain for other problems. One example was that time pressure can lead to a negligence of refactoring which again can lead to technical debt and a decrease in quality. In our case time pressure did force us to prioritize in a different way, which is good to learn because being flexible and intentional as a programmer is useful. We started early on the reports which really helped us, and meeting three times a week was useful. It was both important to deliver minimum value products at the presentations and also deliver on the requirements of the reports. Having deadlines helped us to work efficiently and consistently. Delivering before the deadlines was sometimes challenging. Even though the requirements of the assignments were high, our group managed to reduce stress by working as a team and sharing the workload. While the paper of Meckenstock shows us that deadlines is major challenge to quality of delivered products, it is also important to emphasize that deadlines help us to grow and be efficient as developers.

Technical Debt, Refactoring and Future Proofing

When we developed this project, the technical debt we found ourselves accumulating the most was documentation debt, as defined in [12]. We agree with the paper that our debt was caused mostly by schedule pressure, but also possibly due to non-systematic verification of quality. Whereas we thought testing to be quite important, no one really looked for documentation when reviewing code, and documenting the code was never put on the to-do list, even if it was part of our definition of done.

There were also instances of test debt, especially when we were close to a deadline and wanted to finish a task on the sprint backlog before presenting the sprint retrospect. This debt was mostly cleared up at the start of the next sprint, but this sort of artificially finishing tasks for the "looks" of it could negatively impact work routine and structure, causing technical debt along the way.

Another thing we discussed, and chose to categorize as structural debt, is our reliance on an external API and Json to access course and student information from the universities. As discussed in an example in class, languages like Json can become outdated suddenly, and the whole system would need to be reprogrammed with a new one. Alternatively, the API were reliant on could release a new version, or the universities could choose to start using other platforms, leaving us with a useless product.

As discussed in the section on Software Processes, we recognize the impact refactoring would have on the maintainability of our program. Not all of the things mentioned in the discussion here could be solved by doing refactoring at this time, like the structural challenges, but refactoring does help us break down the program into smaller, more readable parts. In the long term, refactoring would become very important for the maintenance and effectivity of our program.

When it comes to architectural debt, we found ourselves struggling with the combination of our frontend and backend frameworks. This is an inherent architectural debt that stemmed from our inexperience. It is hard to tell if there are any other architectural challenges, as we did not get the chance to work on the project long enough to discover them.

We have also not identified any technology gaps, although there probably are gaps we are not yet aware of. The choices we made for technology are rather modular and we therefore might be able to migrate technologies without much trouble.

The way we have built our system should facilitate for scaling, for example when we start accommodating the page for other institutions other than the two we focused on during the project.

Use of AI in the Project

In our project, AI tools was used on a small scale, but contributed significantly in improving our workflow. Specifically, we integrated AI tools to assist with writing tests, styling our frontend using CSS, and creating a lightweight Python server to test functionalities such as login. AI was particularly useful when we were dealing with parts of the codebase we were less familiar with, allowing us to quickly generate examples and templates we could get inspiration from. One important feature of modern Integrated Development Environments (IDEs) is that they commonly provide the option to use AI for code completion, which most of us probably use without much concern.

Our use of AI aligns with the finding in [7]. The survey highlights that large language models (LLMs) can boost developer productivity, lower the barrier for less experienced programmers, and accelerate software development processes. In our case, AI reduced the time needed to write boilerplate code and improved our understanding of testing practices. The survey also states, one of the strengths of LLMs is their ability to support code generation.

Our experience of boosted productivity is also backed up by this article [20]. The findings in this article explains how development and maintenance of software is undergoing an accelerated transformation thanks to the introduction of large language models. Further on, the article mentions recent studies which results show that the developers productivity can grow as much as 20-50% in repetitive tasks, updates, and first prototypes, to name a few.

Overall, while AI was not something which we relied heavily upon, it proved its worth by making certain tasks more efficient and helped us learn faster. Based on both our experience and the evidence presented in the survey, we believe that AI tools can provide a benefit to software development projects, even when used on a small scale.

Research

Before starting the project, we were introduced to Design Science research as a research framework [5]. It is a framework which structures the process of developing software that solves a problem, in a given domain, with an existing base of knowledge about the and an iterative cycle of development and evaluation. There are some aspects of the Design Science framework that are well known to already practitioners of software development. The iterative cycle where the software artifact is tested and validated with users is an example of such an aspect. But the aspect of contributing back to the knowledge base is often generally not a part of the regular work of creating software. Therefore, only some aspects of Design Science would be useful for creating software. Other aspects of Design Science are more relevant for doing research involving software.

As a framework to follow for work with a master's thesis, there could be several challenges with using Design Science as a framework. Evaluation of the software artifact would be challenging. Often the software is very niche, and therefore evaluation of the software would become very specific to the few users. Or, the evaluation may be too costly or take too long to do in an appropriate manner. This can then in turn diminish the contribution of knowledge. Unfamiliarity with the domain or knowledge base could cause problems with the use of time versus the time used developing.

Design science focuses on making a solution to a problem, but there are other ways to do research in software engineering. A different approach is doing research that tries to explain a phenomenon rather than create a solution to a problem. The approach described in [22] lays out how one can do research around the making of software and the processes that are used while making software. The research methods described in [22] focus more on the explanatory side of software research and not the solution aspect of Design Science. The paper [22] describes approaches that try to understand and investigate, where the goal is more validating a claim, theory, or establish a truth, especially with regards to creation of software. They are based more on observations and evidence from a real-world setting or more theoretical approaches such as simulations and formal theory. These then point towards either disproving a hypothesis or verifying a theory. The knowledge gained leads not towards a solution but adds to the general knowledge base and theory.

Solution-seeking research prioritizes the creation and evaluation of a software artifact to

solve a practical problem, with contributions to knowledge being a crucial extended outcome. In contrast, the phenomenon explaining research primarily aims to understand, explain, or predict the phenomena, generating findings, theories, or hypotheses as its main outcome.

Evaluation of papers

Paper [3] deals with the effects of pair programming. It is a systematic literature review of existing studies comparing the effectiveness of pair programming. The focus is on understanding the general effects of pair programming on project duration, effort, and quality. It tries to summarize the studies in an appropriate way, giving relative weight to different studies based on their sizes, which is sound. The paper has chosen good output parameters for their study, duration, quality, and effort. The papers weakness comes from the studies that it is looking at and the differing definition and quantification of the outcome variables. Quality is measured differently in the papers reviewed, weakening the ability to compare. The quality and generalization of a [3] are fundamentally limited by the quality and characteristics of the primary studies it reviews.

The paper [16] is a case study that looks at the effects of refactoring. The article looks at the impact of explicit refactoring activities on software quality. With the checklist from [25] as an evaluation framework for the article, we claim that the article does not meet step 3 of the checklist. The study does not mention the use of multiple methods of data collection. The paper only collects data in one way with an IDE-based tool that looks at one activity, the coding, and the code produced. The paper [25] mentions that a case study requires the use of multiple sources of evidence that require different methods of data collection facilitating triangulation of results. The size of the small team under review is also a concern. Overall, the method used in the paper seems sound and consistent with the outcome, but it has some factors that weaken its claims.

The paper [8] presents a controlled experiment investigating the effect of task granularity on the software quality produced by novice developers using Test-Driven Development (TDD). While the experiment design and method are good, our main point of contention is the lack of a control group not using TDD. A major improvement of the study would be to include a control group that also has different levels of granularity, but does not do TDD. The paper references a previous study which does have a control group while looking at the effects of TDD, however the task granularity was not part of the paper mentioned by [8]. The paper states that the

experiment concerns only the granularity in the context of TDD. It leaves looking at the effect of task granularity with or without TDD to further research.

Final Reflection - Designing Our Ideal Software Development Process

Throughout the project we used various processes to develop our app. We will now discuss our experience and come up with an ideal software process. The following reflections all point towards how we would have adhered to, or diverted from the software development processes utilized over the course of the project.

What went well?

We found it very useful to meet at fixed times three times a week as mentioned in the paragraph Time pressure. We already discussed how time pressure can be a big challenge. Through frequent meetings we achieved constant progress, planning and communication throughout the weeks. When there is a set deadline ahead of time, it can be tempting to procrastinate until the last week and deliver a rushed product with less quality. Having constant meetings helped us to keep up with the workload and coordinate tasks together smoothly,

We followed a flexible plan that we adjusted along the way. It helped us to work as a team well, and we could discuss technical problems. We could also get more ideas when we chose to brainstorm together, which could give us better results. Our group benefited both from having a plan and being flexible throughout the project.

The use of Scrum and Kanban board was very useful because it helped us plan tasks and divide the work between us, We explained our method in detail in paragraph Software Processes. As mentioned earlier, we combined different methods of different software practices. The sprint planning meetings helped us to align expected progress. We could discuss tasks and find out if people had different expectations. The project had a variety of tasks. The methods we used allowed us to choose tasks we would like to work with, which gave us extra motivation to work with the project. We appreciated being able to choose both which project to work with and which parts of the project we could develop.

We did additionally find interviews to be a critical part of our pre-study and one of the most important ways we collected requirements for our application. Given that we were to work on a project again from scratch, it is something we would view as critical for a correct assessment of the efficacy of our idea in addition to gaining a good understanding of the problem space and the project's requirements.

What we would've done differently

To improve the project's structure and management, We could have used more defined roles, such as project leader, scrum leader, architect and designer. Assigning these specific roles would have clarified responsibilities exceeding just general development tasks, ensuring that every critical aspect of the project had a clearly accountable person. This structure would probably have prevented gaps in responsibility and improved overall coordination.

We could have applied Test-Driven Development differently, for example, prioritizing complex, critical parts of the system, while avoiding TDD for low complexity components where the cost of writing tests clearly outweighs the benefits. This approach would have allowed us to maintain high code quality where it matters the most, while reducing overhead for simple, low risk parts of the system.

We experienced a significant amount of overhead from sprint planning activities such as estimation of effort and story points. See the section on Effort estimation. Throughout our development process these measurements served little utility and consumed a significant amount of time within each sprint planning meeting. We thus did not find them very useful, and mainly an unnecessary drain on our time. They might have been more useful if they had a standardized definition and we had a good amount of experience with. Though in our opinion and with our time constraints, we would rather not utilize these tools.

Tracking user stories in our product backlog was also something we found extraneous. To us it had little utility in this place and only consumed more time during our sprint planning meetings and making the product backlog less clear and precise. Generally we found that we prefer working with features, tasks and requirements directly rather than obscuring them behind user stories. We do still believe in their utility as tools for defining requirements and critically reflecting on the direction of our product, but we do not think that they explicitly belong in the product backlog.

When considering our chosen software tools for the project, we would've likely chosen a different architecture. This isn't precisely related to the process, but something we thought valuable to mention. Given our time constraints; if we were to do our project over, we would've likely chosen an integrated front and back end framework. Something akin to the Svelte-kit framework, or utilized some of the dedicated front end features of Springboot. We believe this could've saved us a significant amount of time, as choosing a more unified framework would've saved significant amount of time from integrating these different tools. There is an argument

to be made that the team should rather argue for the use of specific tools for specific domains based on the requirements of the project. Thus choosing a more simplistic combination of tools unless we find that our requirements specifically call for a different approach. This would've served us to be able to deliver a more complete product to our stakeholders within a more reasonable time frame.

Evaluation

Given these changes to our original process, how could we make sure that this new development process is better than our previous one? We could adopt a similar approach to the one from the article *On the Effectiveness of Pair Programming*[3], where we focus on the two primary metrics of effort (time to complete) and quality (quality of final product). Where groups are assigned a specific task and made to complete the same task following different methods. For our domain this would require a rather large stretch of time and commitment, as well as a reasonable amount of participants. Though it'd likely give us some substantial results. Though it should be mentioned that our suggestions are colored by our own personal experiences and preferences and the efficacy might differ substantially between our group and others for that reason.

If we were to swap to another process internally and either continued the project or started another project. After some time we could conduct an internal survey and possibly a thematic review of these surveys to draw a conclusion based on this. The issue with this approach would be that we might not be able to use the same metrics of effort and quality as our previous experience might bias the result. Although a qualitative approach such as a thematic review of an internal survey as mentioned could still give us valuable feedback.

Appendix A

Personas

Per Ulrik (25), student at UIB and HVL

"I am tired of being logged out of HVL instructure because I logged in at MittUib".

Demographic profile:

Personal Background:

Gender: Male

Age: 25 years old

Education: Bachelor in CompSci, first year master student in Informatics integrated between UiB and HvL

Persona group: Student with multiple places of study

Family status: Single

Professional Background:

Professional occupation: Group leader at the University

Income level: 40k / year + student loan

Work experience: Summer internship in an IT consultancy, part time store clerk

User Environment:

Location: At the library, study hall, at home, in class, on the go

Devices: Phone, laptop

Psychographics:

Attitudes: Puts worth in doing his best, positive attitude towards university and homework, likes getting things done fast, be productive

Interests: Software Developing, traveling

Motivations: Wants to be a software developer, wants to have good grades, wants a well paid job after University

Pain-points: Wasting time, having to log in over and over again, lack of overview, switching between platforms.

End goal:

Logging in once and having access to everything I need to get a productive day of studying done.

Scenario:

"I often try to plan my study sessions on the way to the University. I try logging in to the learning platform of one institution, but sometimes I log in with the wrong account and I am locked out of the platform. I find this very annoying, as I have to close the browser and start over. As a master student, I have a lot of things I need to get done, and want to do so as painlessly as possible. If I'm using my computer, I can set up isolation of cookies in the browser to avoid this, but it is more difficult on my phone..."

Mikkel Mars (22), student at UiB and NHH

"Swapping between the canvas pages of UiB and NHH is really confusing".

Demographic profile:

Personal Background:

Gender: Male

Age: 22 years old

Education: Third year of bachelor in Economics

Persona group: Student with multiple places of study

Family status: Single

Professional Background:

Professional occupation: Part time barista at coffee shop

Income level: 60k / year + student loan

Work experience: Part time working on marketing for a small independent flower shop, Part time at a pet store

User Environment:

Location: At the library, study hall, at home, in class, on the go, at café

Devices: Phone, laptop

Psychographics:

Attitudes: Views university as a way of getting a good job. Doesn't care too much about computers, just views them as tools to do their work. Enjoy getting things done so they can focus on other things.

Interests: Skiing, psychology, branding and artisan beer

Motivations: Wants to work efficiently so he can enjoy time with friends and get a prestigious job

Pain-points: Doesn't understand canvas very well. Frequently forgets their passwords and has to keep checking the notes on their phone for the passwords. Often forgets to check different sections of each canvas page, with exception of announcements.

End goal:

A way to view all the necessary information without wasting my time finding login information.

Scenario:

"I want to check any news on my courses on the go or between classes. But it often becomes a struggle as I get logged out and can't remember my passwords. Sometimes the login page just gives me a weird message about the wrong domain and I don't understand why."

Janne Juipiter (36), lecturer at UiA

"My students keep missing important information."

Demographic profile:

Personal Background:

Gender: Female

Age: 36 years old

Education: PhD in Linguistics

Persona group: Lecturer and course coordinator

Family status: Married

Professional Background:

Professional occupation: Professor at UiA

Income level: 700k / year

Work experience: Part time foreign language teacher, language consultant for newspaper, part time flower store clerk

User Environment:

Location: At the office, at home, commuting

Psychographics:

Attitudes: Enjoys teaching, gets annoyed when students miss information

Interests: Tea, traveling, learning languages, hiking

Motivations: Wants to be a good teacher for her students and wants to be viewed as an acclaimed linguist

Pain-points: Isn't great at organizing canvas pages, isn't sure which parts of the page students use frequently, students frequently miss important information

End goal:

A better way of communicating important information to the students

Scenario:

"Janne sits in her office, sipping her favorite cup of green tea while preparing the next week's lecture materials. She carefully uploads the key points, deadlines, and reading materials onto Canvas, but she knows from experience that many students will still miss something important. A few days later, she receives yet another email from a student asking about a deadline she had already posted. Frustrated, she sighs and wonders if there's a better way to ensure that students actually see and engage with the crucial information. She considers rearranging the course page but isn't sure what sections student check most often. Maybe there's a smarter way to highlight important updates.. something more intuitive, something that ensures her students stay informed without her having to chase them down."

Sofia Nakamura (24), primary enrolled at NHH with select courses at UiB.

"The calendar should show all my lectures from both institutions."

Demographic Profile:

Personal Background:

Gender: Female

Age: 24

Education: 3rd year international student in Bachelor in Business.

Economics, and Data Science at NHH. Sofia is a full-time student at NHH, with select courses at UiB as a part of joint-degree program.

Persona group: International student

Family status: Single

Professional Background:

Professional occupation: Part time Data Analytics at a fintech startup. Youtuber.

Income level: 80k / year + student loan

Work experience: Summer internship in at fintech startup.

User Environment:

Location: Library, cafes, study hall, class.

Devices: Phone, laptop.

Psychographics:

Attitudes: Enjoy learning, hanging out with friends, not very patient

Interests: Financial analytics and Investments, Entrepreneurship and startups, travelling, spending time at café with friends. Enjoys making content for YouTube about her analytics about market trends and how to create successful startups.

Motivations: Career driven. Wants a high paying job so she can support her elderly parents.

Pain-points: Does not speak the native language. Managing courses across two institutions. Balancing studies with work experience.

End goal:

Important information for her degree such as announcements, time sensitive exercises or quizzes can be found at one location on her learning platform.

Scenario:

"I want to be able to read important information regarding my courses at one location. The calendar should show all my lectures from both institutions."

Appendix B

Mock ups

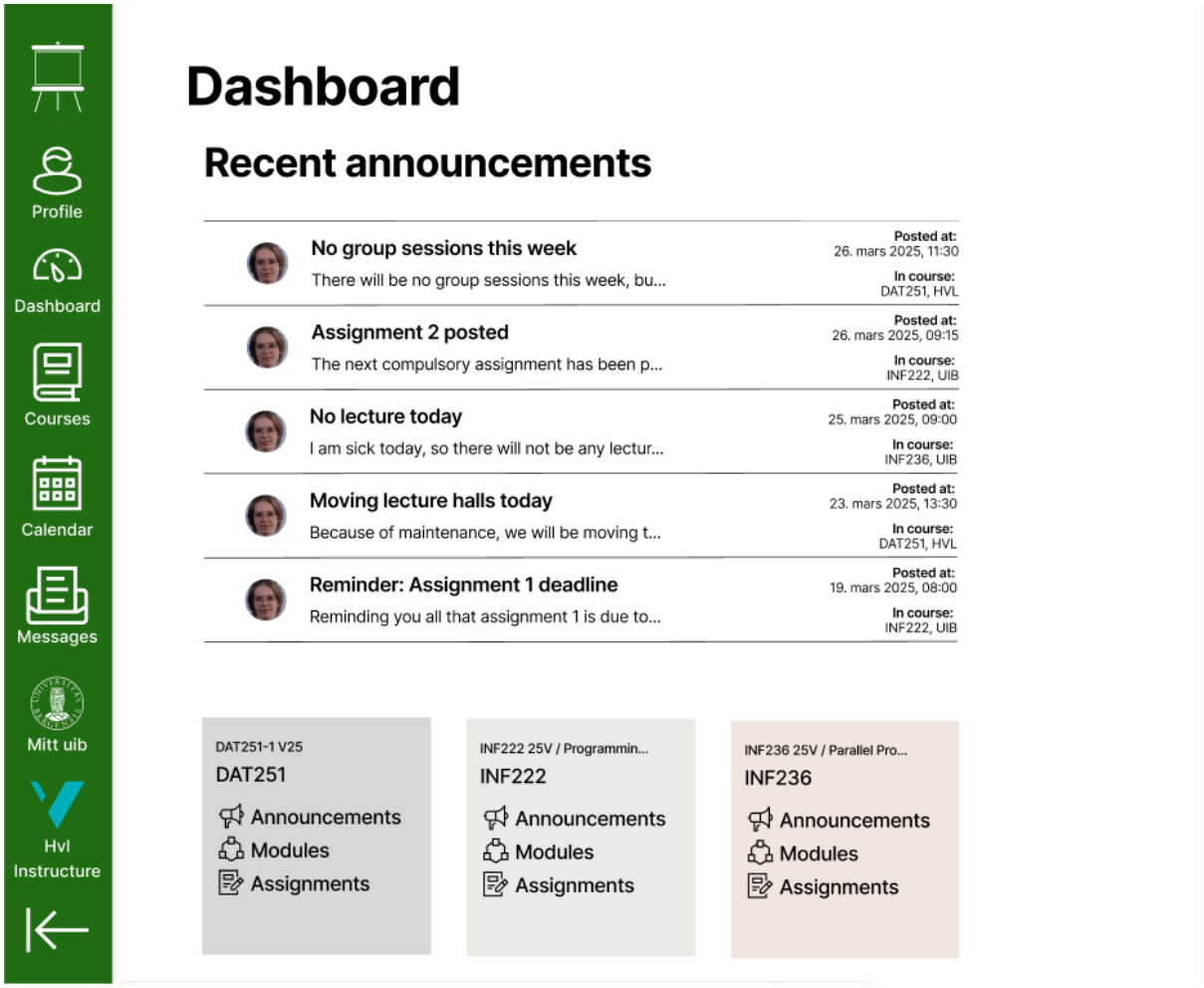


Figure B.1: Mock up showing the dashboard of the web page

Figure B.1 is based on previous mock-ups we made by editing screen shots of the existing mitt.uib.no and hvl.instructure.com. You can see these in figure B.2 and B.3.

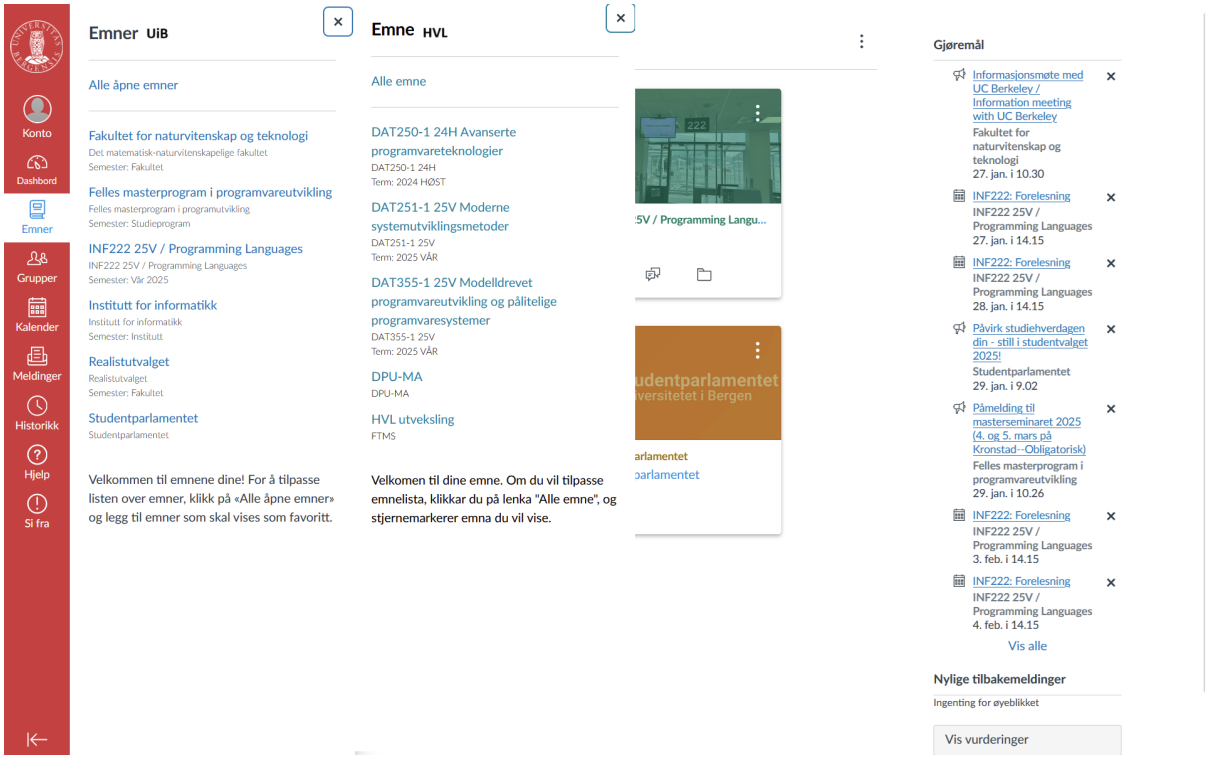


Figure B.2: Augmented screenshot of the mitt.uib.no website with HVL courses added for visualizing purposes.

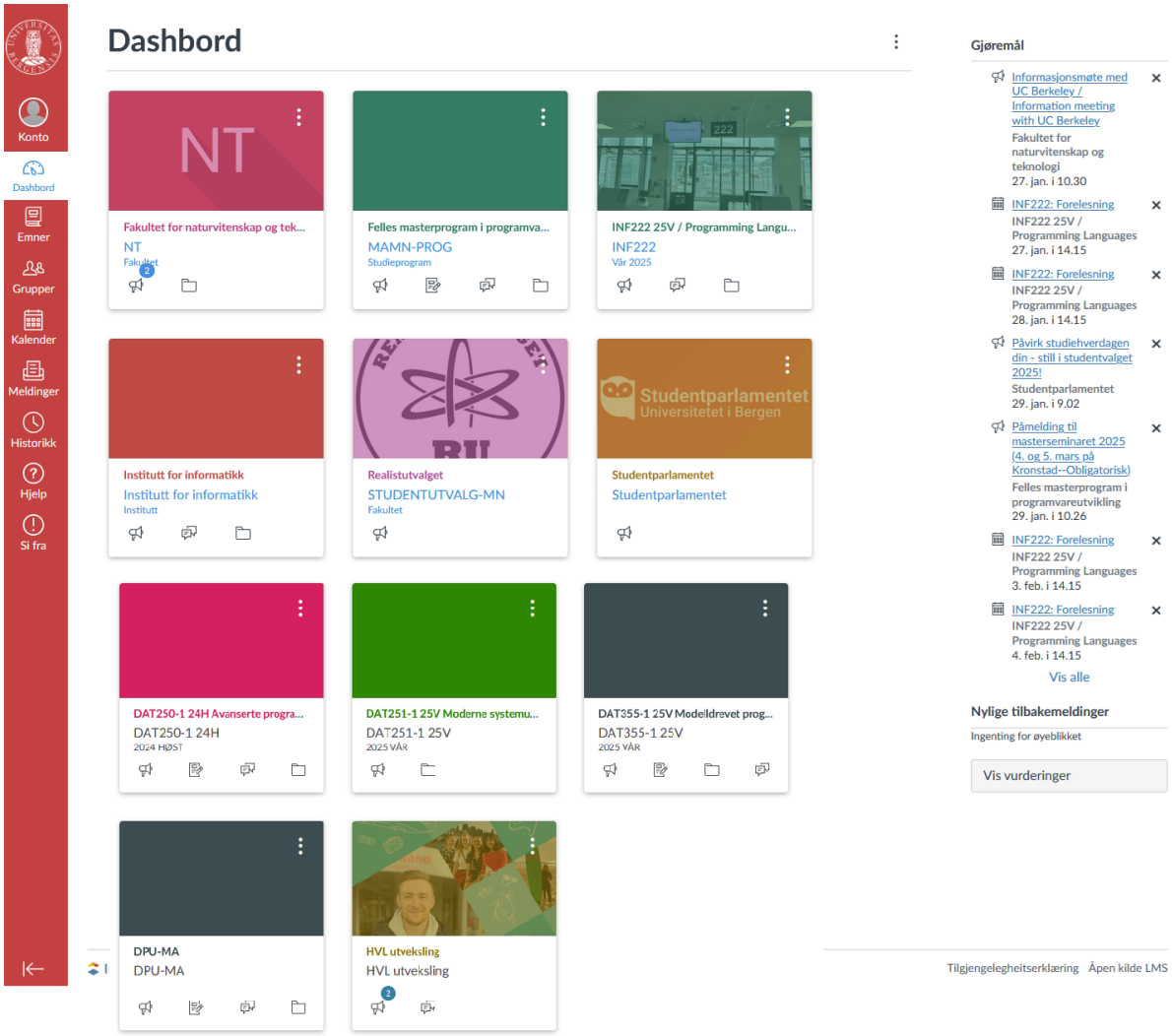


Figure B.3: Augmented screenshot of the mitt.uib.no website with HVL courses added for visualizing purposes.

Appendix C

SAF Toolkit

We are looking to understand the different concerns of the application. We used the SAF toolkit for this, and divided the quality concerns into the 4 sustainability dimensions: social, technical, environmental and economic. Then we brainstormed and sorted the concerns to get a better overview and understanding. This process can be seen in figure C.1, where blue is technical, green is environmental, orange is social and red is economic. Concerns used in the decision map are not depicted here, as these images were taken after applying the SAF toolkit and thus moving these concerns to the Desicion Map.

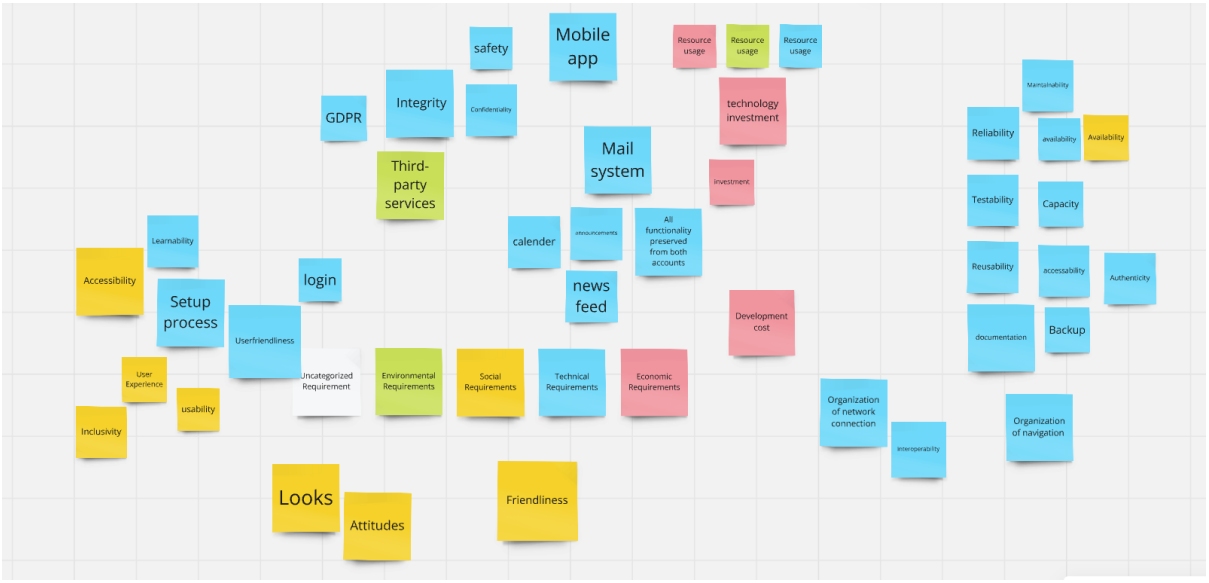


Figure C.1: Brainstorming of concerns related to the application's domain

We then decided what we thought would be the most important concerns and sorted these in a Decision Map (DM). According to the SAF Toolkit, the Decision Map is ordered by the order of impacts: immediate, enabling and systemic. We divided them into the following categories:

Immediate impact, enabling impact and systemic impact. These categories are used to get an overview over potential short term, mid-term and long term positive and negative outcomes. The arrows are displaying effects, where green signifies a positive effect, red signifies negative effects and black is undecided. Figure C.2 shows the decision map.

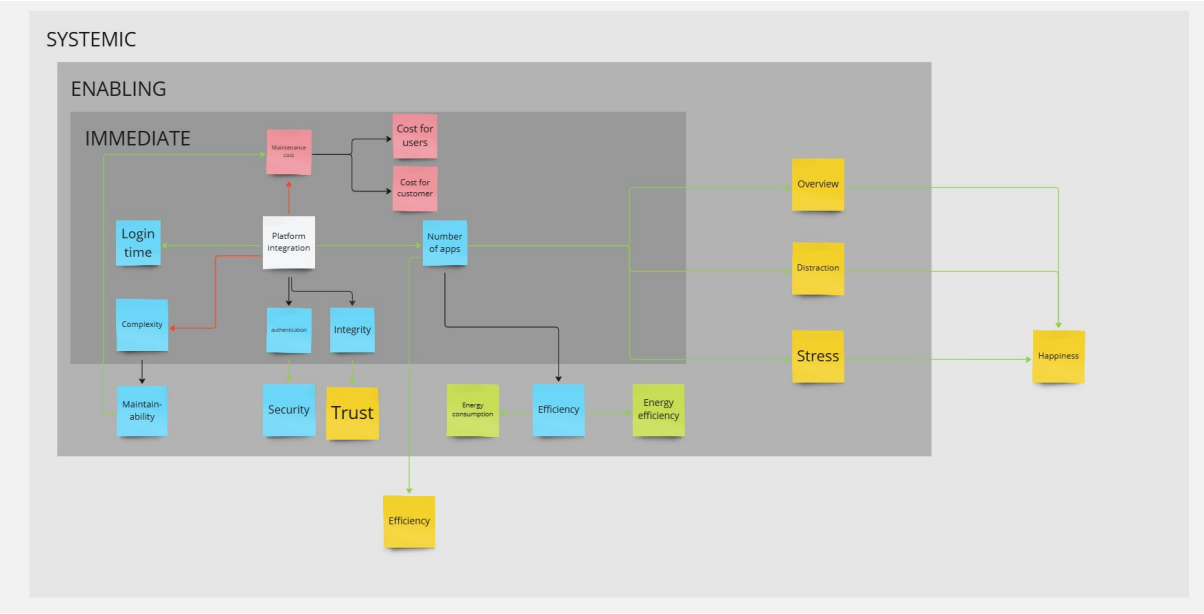


Figure C.2: Concerns organized in a decision map

The next step in the SAF Toolkit was to create a Sustainability Quality Model (SQ Model). We included every concern we had in the DM and defined the concerns in the context of our project, as well as categorized them into the appropriate sustainability dimensions and gave each concern a metric. The SQ Model is sown in figure C.3.

From these different concerns, we then agreed upon four main concerns that we thought would be the most important. These are our Sustainability goals:

- min(login time)
- max(overview)
- max(integrity)
- min(cost for users)

We then used these to make KPI models. These models are useful to see how we can meet our goals. Figure C.4 show the KPI model we made. This is possible to adapt later on as we work with the project.

QUALITY ATTRIBUTE	DEFINITION	REFERENCES	TEC	ENV	ECO	SOC	METRIC/UNIT
Cost for users	The price a user must pay to be able to use the platform				x		Norwegian kroner (*kr*)
Cost for customer	The price a customer must pay to use the platform				x		Norwegian kroner (*kr*)
Maintenance cost	The price to keep the platform running smoothly				x		Norwegian kroner (*kr*)
Complexity	The amount of interacting components	Wikipedia.org	x				Number of software components
Login time	Time spent on logging in per week		x				Duration in *min/week*
Efficiency	The time a user spends to get to the desired information					x	Duration in *s*
Overview	The amount of information available					x	Comparison of platform data vs source (coverage in %)
Authentication	Only authorized users can access classified information		x				Amount of leaks and security breaches
Happiness	The feeling that everything works well					x	Survey responses (Likert scale)
Energy efficiency	Energy usage of the platform per unit of work			x			Efficiency in Watt (W)
Efficiency	The time it takes from a request is made until the response is given		x				Duration in *s*
Trust	One can rely on the information provided					x	Survey responses (Likert scale)
User experience	How much the user likes using the app					x	Survey responses (Likert scale)
Security	Protection from (potential) harm and unwanted coercion	Wikipedia.org	x				Amount of attacks stopped
Disctraction	Number of interruptions when pursuing a specific task					x	Amount in natural numbers
Energy consumption	Amount of electricity used to run the application alltogehter			x			Electricity in kwH
Integrity	The accuracy, completeness, and quality of data as it's maintained over time	Hardvard Business School Online	x				Comparison of platform data vs source (similarity in %)
Number of apps	The number of apps needed to perform a given task		x				Amount in natural numbers
Maintainability	Time from a new feature/fix is introduced until it is ready to be deployed		x				Duration in *days*

Figure C.3: Sustainability Quality Model (SQ Model).

KPI Name	Login Time			
Goal	**Critical Success Factor**	**Key Performance Indicator**	**Metric(s)**	**Measure(s)**
Reduce the amount of time used to log in	Logging in only once per workday	Number of times needed to logging	Time spent logging in (minutes) in a certain period / ammount of weeks in the same period	Ammount of time spent during the login process
		Target / **Action**		
		1 per workday / Adapt the login structure		

KPI Name	Integrity			
Goal	**Critical Success Factor**	**Key Performance Indicator**	**Metric(s)**	**Measure(s)**
Provide security for the application	Removing as many as possible security holes	Attempted attacks prevented	Amount of attacks prevented in a given time	Amount of attacks prevented
		Target / **Action**		
		As high as possible attempted attacks / - Hire a security specialist		

KPI Name	Overview			
Goal	**Critical Success Factor**	**Key Performance Indicator**	**Metric(s)**	**Measure(s)**
Maximizing the amount of relevant information available	Reducing confusion and time spent to find information	Amount of complaints about the lack of overview	Amount in natural numbers	Complaints
		Target / **Action**		
		0 / - Redesign UI - User interviews		

KPI Name	Cost for users			
Goal	**Critical Success Factor**	**Key Performance Indicator**	**Metric(s)**	**Measure(s)**
Reduce cost for users to zero	Reducing budget posts that are reliant on user cost	Sign up fee	Norwegian kroner	Cost for each signup for students
		Target / **Action**		
		0 kr / - Increase customer price - Find sponsors		

Figure C.4: Temporary KPI models for evaluating sustainability goals.

Appendix D

Refactoring

One refactoring method we used a lot was Extract Method. Mostly when using this method, we had identified the code smells Duplicated Code and Long Method. In one class, we had several methods that were doing requests against an external api. Here, we extracted the setting of headers for the request to its own method, as these should be the same for all requests. This method is illustrated in figure D.1 We also extracted the method for getting the user's authorization token. When requesting announcements from the api, the id of the associated courses must be provided as context codes in the format "course_{id}". To avoid a long method where this was needed, the conversion from course id to context code was extracted.

```
3 usages   tyraf
private HttpEntity<String> buildRequest(String token) {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    headers.setAccept(Collections.singletonList(MediaType.APPLICATION_JSON));
    headers.set("Authorization", "Bearer " + token);
    return new HttpEntity<>(headers);
}
```

Figure D.1: buildRequest extracted method

We had two methods that retrieved announcements. One would get the announcements for a specified institution and a specified list of courses, while another method would get the announcements for all institutions and all courses. The preparation for each method was different, but the part where they sent a request to the api was the same. We thus extracted this part to its own private method. This part turned out to still be quite a long method, and we ended up extracting part of it to its own method yet again. We thus separated the part where we made

the request and the part where the result was parsed and organized. This can be seen in figure D.2.

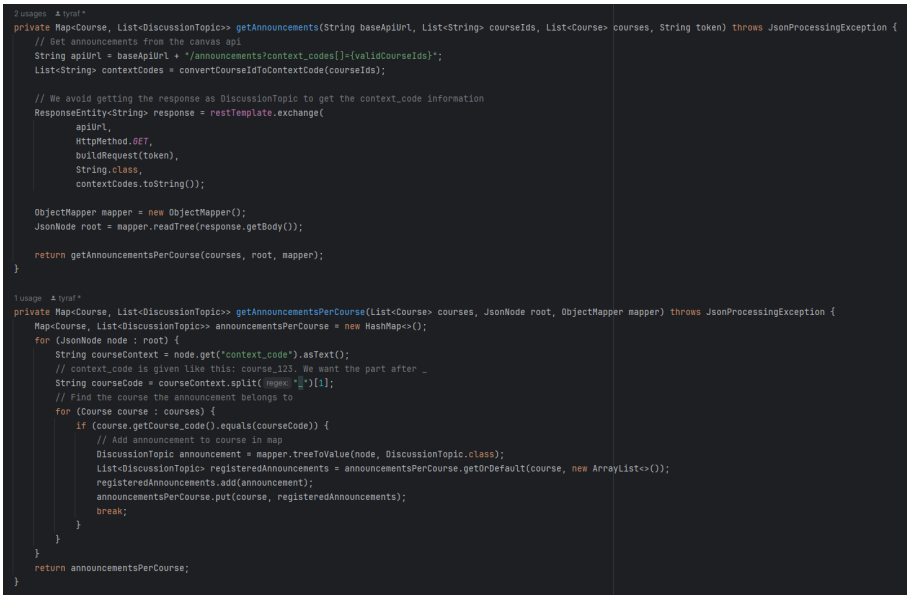


Figure D.2: getAnnoucnements extracted method and getAnnoucnementsPerCourse extracted method

For a large part of this iteration, we created our program with the assumption that we were only going to accommodate two Canvas institutions - The University of Bergen and The Western Norway University of Applied Sciences. Thus, it was easy to just create an if-else statement when we wanted to base behaviors on which institution we were operating with. This happens multiple places, and one example of this duplication can be seen in figure D.3. This is a code smell, as our platform potentially should host more institutions and we would then have to expand every single if-else statement when expanding our hosting. To combat this, we used the refactoring method Extract Class.

It was decided that the institution information should be stored in the database of our

Figure D.3: getBaseApiUrl method before refactoring

application. This way, administrators of the application could update this information through a website admin interface. The institution object can be seen in figure D.4. Getter and setter methods have been omitted. A repository and service was created to manage storing and retrieving the institution information from the database. The method in figure D.3 was changed accordingly to the one shown in figure D.5. After refactoring, some tests failed because the class CouseService that the 'getBaseApiUrl' was in now needed one more argument in the constructor. But the method itself did not change behavior, and we thus count this as a refactor.

```
19 usages  tyraf
@Entity
@Table(name = "institution")
public class Institution {

    2 usages
    @Column(unique = true, nullable = false)
    private String fullName;

    @Id
    @Column(unique = true, nullable = false)
    private String shortName;

    2 usages
    @Column(unique = true, nullable = false)
    private String apiUrl;

    tyraf
    public Institution() {}
```

Figure D.4: Institution object

```
2 usages  tyraf
private String getBaseApiUrl(String institution) {
    String baseApiUrl = institutionService.getApiUrlByShortName(institution);
    if (baseApiUrl == null) {
        throw new HttpClientErrorException(HttpStatus.NOT_FOUND, "No such institution found" + institution);
    }
    return baseApiUrl;
}
```

Figure D.5: getBaseApiUrl method after refactoring

These refactorings helped us adhere to the single responsibility principle, which was a design principle we strived towards throughout the production.

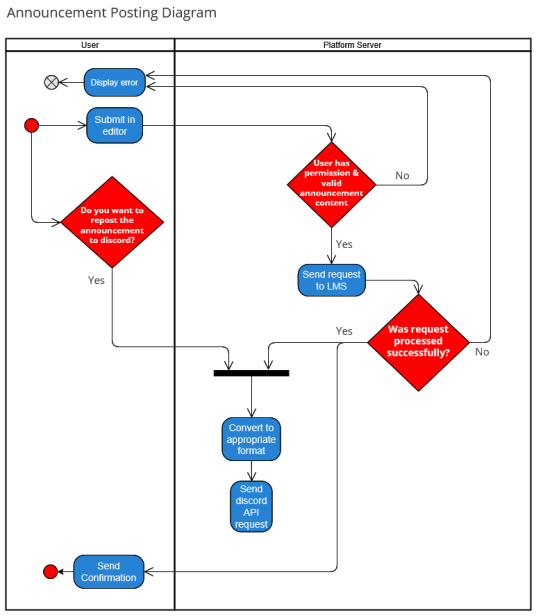
Appendix E

Models

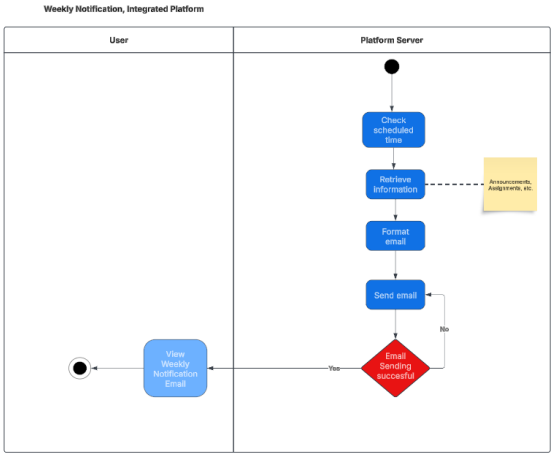
This appendix mainly includes artifacts from the development process. Mainly serves as reference for the main report.

Activity Diagrams

The following are activity diagrams, illustrating the functionality of different features

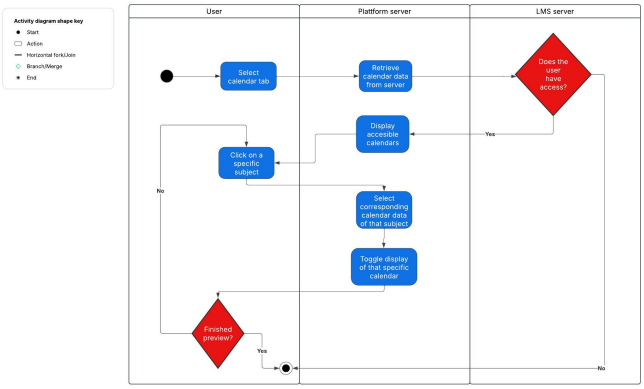


(a) Activity Diagram of Announcement Posting

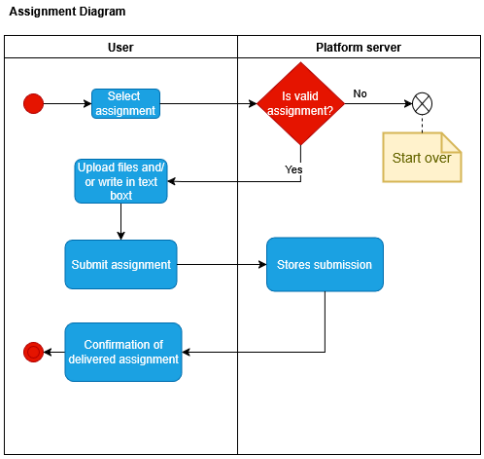


(b) Activity Diagram of Weekly Notifications

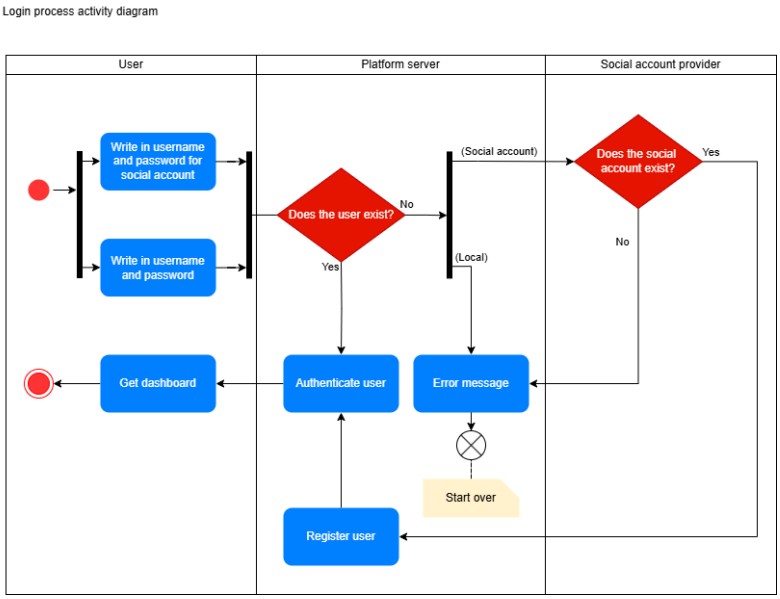
The following are more such activity diagrams



(a) Activity Diagram of Calendar



(b) Activity Diagram of Submitting Assignments



(c) Activity Diagram of User Login

Other Models

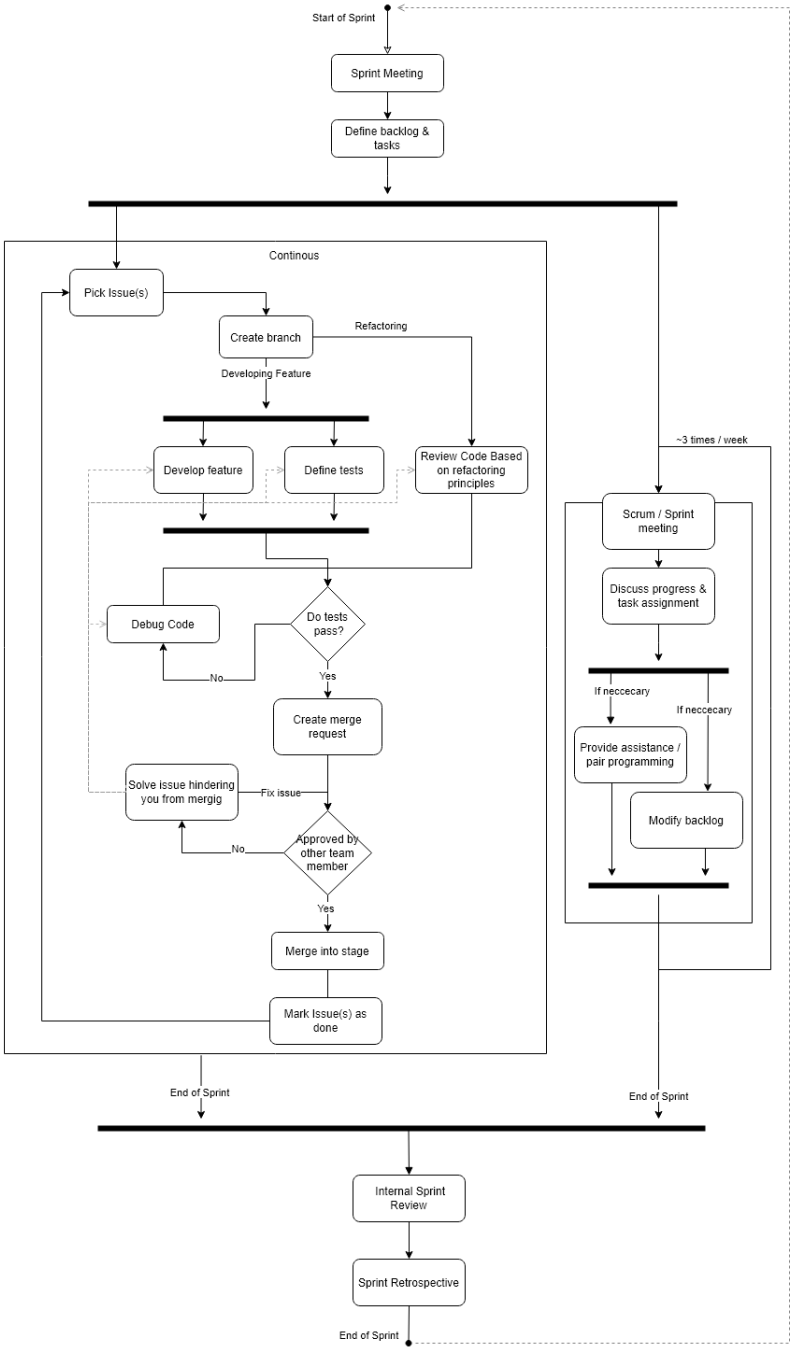


Figure E.2: Activity Diagram, illustrating our development process

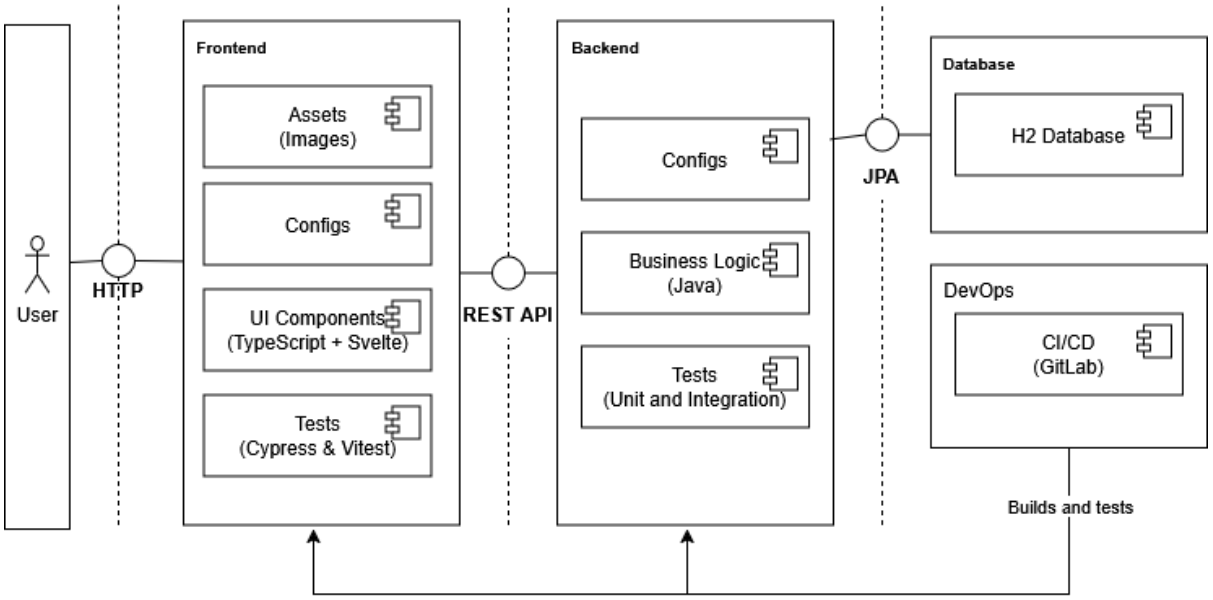


Figure E.3: Model of the architecture

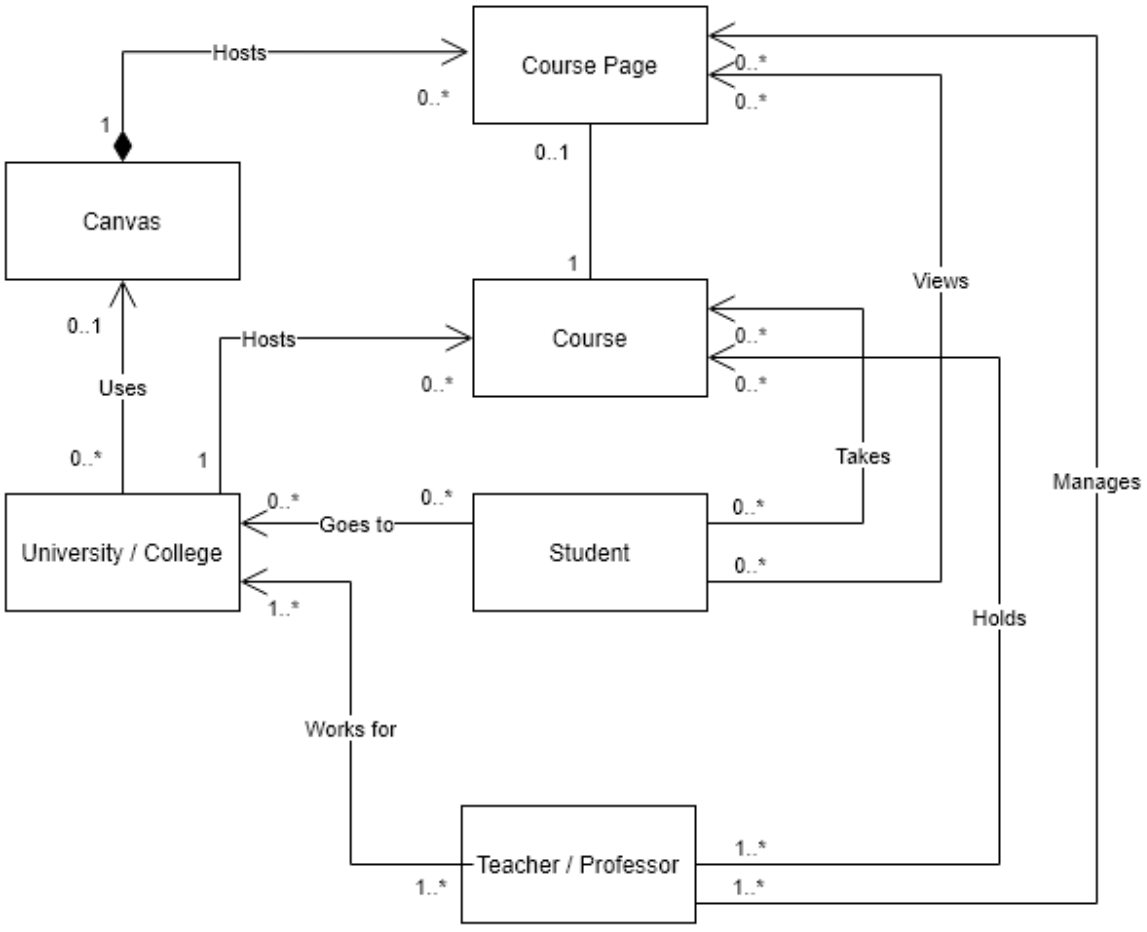


Figure E.4: Domain Model

Appendix F

Software Development Process and Team practices

Our software development process, as seen in figure E.2, generally consists of the basic Scrum structure. Where our work is split into different sprints. Following our sprint startup meeting we concurrently work on issues while conducting sprint stand-up meetings 3 times a week. In our meetings we discuss what we've worked on since our last meeting and what our plans are going forward. This is where we might group up to help each other with issues and/or conduct pair programming. There are also cases where we modify our backlog, most often since new issues have arisen. In parallel we pick issues individually to work on, though this does often happen during our stand-ups. Here we have a standard procedure where we create a new branch for our issue and either develop a new feature, including tests; or conduct refactoring. Then after the tests have passed and we are satisfied, we create a merge request. Wherein only other team members who did not submit the request can approve them. Thereafter we mark our issues as done. Following the end of our sprint, we conduct a short internal sprint review and retrospective.

F.1 Collective Ownership

All members of the project are owners of the code repository hosted on Gitlab. This means that anyone on the team can contribute to the any part of the codebase and is alone with responsibility of any specific section. Any member can add user stories, spikes or tasks to the product backlog, as well as deleting/editing as they see fit - with appropriate notice to the team.

It's working well because of the good communication within the team. By incorporating this open and transparent workflow, we make sure that everyone's ideas and improvements are welcome.

One potential drawback is that not everyone on the team may be fully aware of what others are working on at a given time. Which could lead to unintentional conflicts when modifying the same parts of the codebase. But we try to mitigate this by having regular updates in stand-ups and by tracking task ownership.

F.2 Updating the Product Backlog

When updating the Product backlog and creating the Sprint backlog we made sure to create an Acceptance Criteria, Definition of Ready, and Definition of Done for the most important user stories. We also created it for some of the less important user stories, but we did not prioritize this, as our work needed to be shifted elsewhere because of time constraints.

When creating the user stories we also used Planning Poker to determine the story points we wanted to allocate the different user stories. The use of Planning Poker showed us that our group had quite polarizing views when it came to almost every important user story. This then gave everyone the opportunity to explain why they felt like they chose the amount of story points that they did. This gave the group more understanding about how much work our different user stories will actually require of us.

F.3 Customer Tests and Test-Driven Development (TDD)

We did not conduct users test in the first sprint work of the project due to not having prototype ready (as of time of writing). That is we do not include ourselves when developing first iteration.

We tried doing Test-Driven Development at early stages in the development process, but due to a lack of understanding of the framework, this became almost impossible. As some members had never used the framework before, it was very hard to know what the expected behavior should be, let alone how to write tests for it. With simpler classes that were not tied to the framework, Test-Driven Development worked smoothly and ensured a high test coverage from the get-go. However, it was experienced as a rather slow way of producing code, and with such a clear idea of how the implementation should be, it was difficult to have the patience for writing tests before starting implementation. Often times, the class was created before it was

remembered that tests should be written.

As we became more familiar with the framework and its expected behavior, its clear that TDD can be a highly effective approach to development. The benefits of TDD are much more apparent now: it encourages us to think critically through the edge cases before implementing the solution, which leads to clearer and more maintainable code. TDD allowed us to take a step back from a complicated class and think about what it was we were actually trying to do. It helped us get a clearer overview of the requirements and helped us organize our work.

F.4 Scrum Meetings

We held 3 Scrum meetings a week, where we shared what we had worked on since the last meeting, any pending issues, and what we planned to do until the next meeting. This structure worked quite well, except from the fact that as full-time students, it was not always possible to get the time to do very much between some of the meetings. On the positive side, it gave the group a very nice overview of what was going on and what they should spend their time doing.

F.5 Burndown Chart

The burndown chart for the third iteration can be found in figure F.1. The burndown chart was not helpful to us, as we had frequent meetings updating each other on progress without the need to consult the chart to check if someone was stuck. We were not very good at properly defining what we were currently working on, and perhaps viewed some tasks as one bigger task, where the tasks were worked on simultaneously and were not marked as done until every task was done. This made it difficult to keep the burndown chart updated, and its purpose became purely to answer the task at hand which required such a chart.

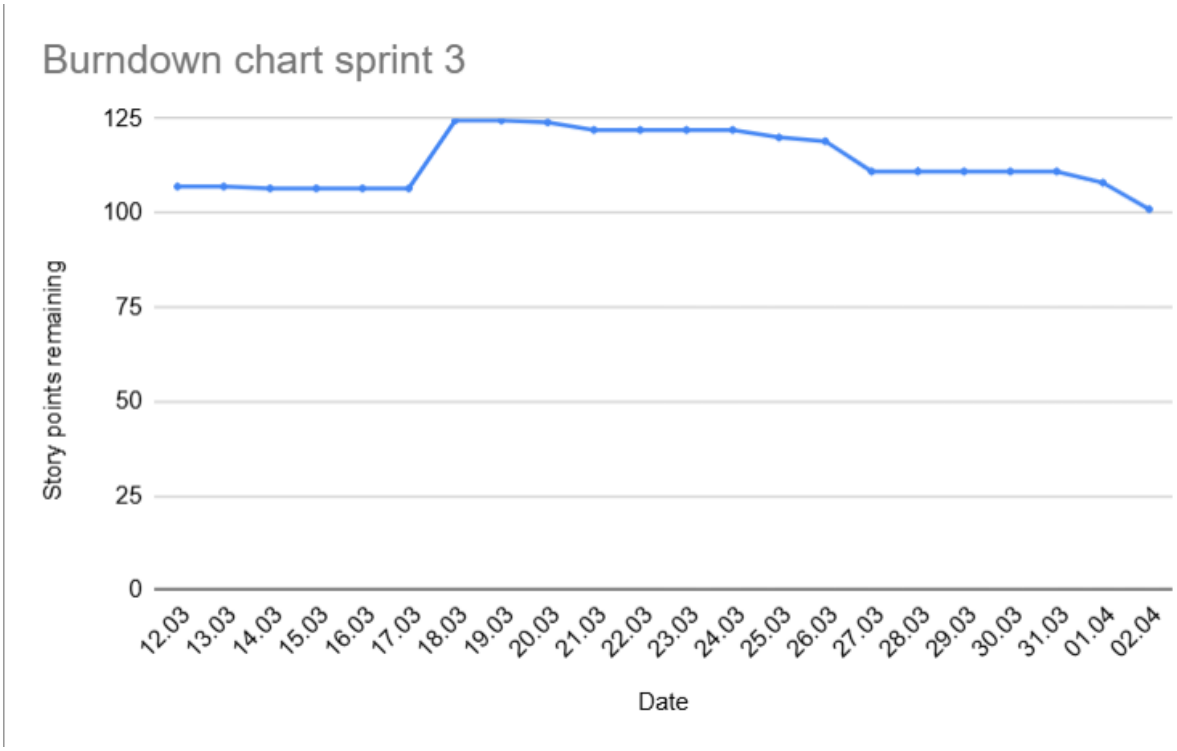


Figure F.1: Burndown chart for the third iteration

Bibliography

[1] Ackermann, R.: Design thinking was supposed to fix the world. where did it go wrong?, https://www.technologyreview.com/2023/02/09/1067821/design-thinking-retrospective-what-went-wrong/

[2] Bencomo, N., Cabot, J., Chechik, M., Cheng, B.H.C., Combemale, B., Wasowski, A., Zschaler, S.: Abstraction engineering. https://doi.org/10.48550/arXiv.2408.14074, http://arxiv.org/abs/2408.14074

[3] Dybå, T., Arisholm, E., Sjoberg, D.I., Hannay, J.E., Shull, F.: Are two heads better than one? on the effectiveness of pair programming **24**(6), 12–15. https://doi.org/10.1109/MS.2007.158, https://ieeexplore.ieee.org/document/4375233/

[4] Heldal, R., Pelliccione, P., Eliasson, U., Lantz, J., Derehag, J., Whittle, J.: Descriptive vs prescriptive models in industry. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems. pp. 216–226. ACM. https://doi.org/10.1145/2976767.2976808, https://dl.acm.org/doi/10.1145/2976767.2976808

[5] Hevner, A., Chatterjee, S.: Design science research in information systems. In: Hevner, A., Chatterjee, S. (eds.) Design Research in Information Systems: Theory and Practice, pp. 9–22. Springer US. https://doi.org/10.1007/978-1-4419-5653-8_2, https://doi.org/10.1007/978-1-4419-5653-8_2

[6] Instructure, Inc: https://www.instructure.com/

[7] Jiang, J., Wang, F., Shen, J., Kim, S., Kim, S.: A survey on large language models for code generation. https://doi.org/10.48550/arXiv.2406.00515, http://arxiv.org/abs/2406.00515

[8] Karac, I., Turhan, B., Juristo, N.: A controlled experiment with novice developers on the impact of task description granularity on software quality in test-driven development **47**(7), 1315–1330. https://doi.org/10.1109/TSE.2019.2920377, https://ieeexplore.ieee.org/document/8727972/

[9] Klunder, J., Hebig, R., Tell, P., Kuhrmann, M., Nakatumba-Nabende, J., Heldal, R., Krusche, S., Fazal-Baqaie, M., Felderer, M., Genero Bocco, M.F., Kupper, S., Licorish, S.A., Lopez, G., McCaffery, F., Ozcan Top, O., Prause, C.R., Prikladnicki, R., Tuzun, E., Pfahl, D., Schneider, K., MacDonell, S.G.: Catching up with method and process practice: An industry-informed baseline for researchers. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). pp. 255–264. IEEE. https://doi.org/10.1109/ICSE-SEIP.2019.00036, https://ieeexplore.ieee.org/document/8804460/

[10] Kramer, J.: Is abstraction the key to computing? **50**(4), 36–42. https://doi.org/10.1145/1232743.1232745, https://dl.acm.org/doi/10.1145/1232743.1232745

[11] Kruchten, P.: What do software architects really do? **81**(12), 2413–2416. https://doi.org/10.1016/j.jss.2008.08.025, https://www.sciencedirect.com/science/article/pii/S0164121208002057

[12] Kruchten, P., Nord, R.L., Ozkaya, I.: Technical debt: From metaphor to theory and practice **29**(6), 18–21. https://doi.org/10.1109/MS.2012.167, http://ieeexplore.ieee.org/document/6336722/

[13] Lago, P., Condori Fernandez, N., Fatima, I., Funke, M., Malavolta, I.: The sustainability assessment framework toolkit: a decade of modeling experience (2024). https://doi.org/10.1007/s10270-024-01230-9, https://doi.org/10.1007/s10270-024-01230-9

[14] Meckenstock, J.N.: Shedding light on the dark side – a systematic literature review of the issues in agile software development methodology use **211**, 111966. https://doi.org/10.1016/j.jss.2024.111966, https://linkinghub.elsevier.com/retrieve/pii/S0164121224000098

[15] Memon, A., Zebao Gao, Bao Nguyen, Dhanda, S., Nickell, E., Siemborski, R., Micco, J.: Taming google-scale continuous testing. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP). pp. 233–242. IEEE. https://doi.org/10.1109/ICSE-SEIP.2017.16, http://ieeexplore.ieee.org/document/7965447/

[16] Moser, R., Abrahamsson, P., Pedrycz, W., Sillitti, A., Succi, G.: A case study on the impact of refactoring on quality and productivity in an agile team. In: Meyer, B., Nawrocki, J.R., Walter, B. (eds.) Balancing Agility and Formalism in Software Engineering, vol. 5082, pp. 252–266. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-85279-7_20, http://link.springer.com/10.1007/978-3-540-85279-7_20, series Title: Lecture Notes in Computer Science

[17] Norsk helsenett: Helsenorge-appen. https://www.helsenorge.no/helsenorge-appen/ (2021), updated 08.04.2021, accessed 05.03.2025

[18] Plattner, H.: An introduction to design thinking: Process guide (2010), https://alnap.org/help-library/resources/an-introduction-to-design-thinking-process-guide/, accessed: 2025-04-29

[19] Potvin, R., Levenberg, J.: Why google stores billions of lines of code in a single repository. Communications of the ACM **59**(7), 78–87. https://doi.org/10.1145/2854146, https://dl.acm.org/doi/10.1145/2854146

[20] Sauvola, J., Tarkoma, S., Klemettinen, M., Riekki, J.: Future of software development with generative ai. Automated Software Engineering (2024). https://doi.org/10.1007/s10515-024-00426-z, https://doi.org/10.1007/s10515-024-00426-z

[21] SIKT - Norwegian Agency for Shared Services in Education and Research: https://www.feide.no/

[22] Stol, K.J., Fitzgerald, B.: The ABC of software engineering research **27**(3), 1–51. https://doi.org/10.1145/3241743, https://dl.acm.org/doi/10.1145/3241743

[23] University of Bergen: UiBtreet - Inngangsporten til UiB. https://treet.uib.no/, accessed 05.03.2025

[24] Waterman, M., Noble, J., Allan, G.: How much up-front? a grounded theory of agile architecture. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. pp. 347–357. IEEE. https://doi.org/10.1109/ICSE.2015.54, http://ieeexplore.ieee.org/document/7194587/

[25] Wohlin, C., Rainer, A.: Is it a case study?—a critical analysis and guidance **192**, 111395. https://doi.org/10.1016/j.jss.2022.111395, https://linkinghub.elsevier.com/retrieve/pii/S0164121222001121