Imperativ och objektorienterad programmeringsmetodik

Projektrapport - HT24

Vapid GC

Författare:
Edvin Sundin
Hampus Toft
Vilhelm Lundberg
Momo Sundbäck Brohult
Filip Hellgren
Rouaa Soukar
Magdalena Byrska

17 januari 2025

Innehåll

1 Participant list

Namn	Email	Aktiva datum
Edvin Sundin	edvin.sundin.3750@student.uu.se	13/12/2024 - 17/1/2025
Hampus Toft	hampus.toft.1883@student.uu.se	13/12/2024 - 17/1/2025
Vilhelm Lundberg	vilhelm.lundberg.2366@student.uu.se	13/12/2024 - 17/1/2025
Filip Hellgren	filip.hellgren.9702@student.uu.se	13/12/2024 - 17/1/2025
Momo Sundbäck Brohult	momo.sundback-brohult.1415@student.uu.se	13/12/2024 - 17/1/2025
Rouaa Soukar	rouaa.soukar.4785@student.uu.se	13/12/2024 - 17/1/2025
Magdalena Byrska	magdalena.byrska.5347@student.uu.se	13/12/2024 - 17/1/2025

Tabell 1: Participant list

2 Quantification

Project start and end date
The project started on the 13th of December 2024, and ended with a seminar on the 17th of January 2025.

Number of sprints, their start and end dates
Sprint 1: December 13 - December 22
Sprint 2: December 27- January 3
Sprint 3: January 3 - January 8
Total number of new lines of C code written excluding tests and preexisting code
3255 total lines of C code (src folder).
Total number of lines of test code
4418 total lines of test code.
Total number of lines of "script code" (e.g., make files, Python scripts for generating test data, etc.)
953 lines of script code (Makefile).
Total number of hours worked by the team.
400 hours.
Total number of git commits
503 total git commits.
Total number of pull requests
78 total pull requests.

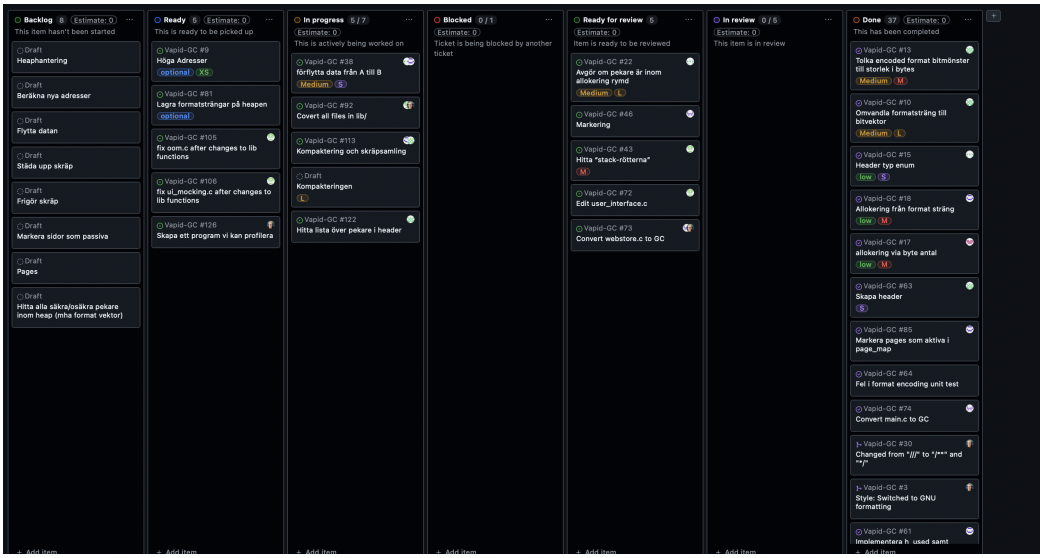
Total number of GitHub issues
50 total GitGub issues.

3 Process

3.1 Inception

In the beginning of the project, we did not do any coding for the first week. Instead, we started reading about different processes and came to the conclusion that we wanted to use a combination of Scrum and Kanban. After making this decision, we approached the project by breaking it down into smaller parts. We used a shared Google document to write down the different features we planned to implement. After this, we assigned roles within the group, allowing the people assigned to the specific roles to have some extra input within that area. We designated a project leader and assigned roles such as "Lead Test engineer", "Lead Documentation engineer" and "Lead Project Report Author".

Further on, our project leader set up a GitHub project containing the different issues we had previously identified in the backlog. Kanban was implemented using Github projects. GitHub projects is similar to a visual board with different columns representing different stages of the workflow. We used Scrum to divide our work into three different sprints, with each sprint being around one week long. Our plan for the first sprint was to study garbage collection and to create the backlog with the different tasks. During the second sprint, we aimed to implement all the foundational functions and for the last sprint we focused on finalizing the compacting part and integrating the program with a previous course assignment. We also began writing the report and planning for the presentation during the second sprint. As part of the process we also planned to have daily meetings over Discord to discuss the progress and the upcoming sprints.



Figur 1: An example of how Kanban was used in GitHub.

3.2 Implementation

As discussed earlier, we implemented a visual board using GitHub projects following the Kanban process, and developed a backlog and a sprint plan using the Scrum method. Some positives with our implementations were that the visual board in GitHub proved to be highly convenient, and provided a clear overview of the different tasks and stages of the workflow. The use of the sprint planning and backlog with Scrum, made it easy to track the process and to set goals for each sprint.

However, our implementation also had some weaknesses. We did not have daily scrums and our sprint plan lacked detailed goals. Also, some team members may have been overloaded with work, so the work balance was not great. If we were to start over tomorrow, we would make a more detailed sprint plan including clear goals with hard deadlines. We would also aim to have daily meetings and try to distribute the work more evenly. One more thing we would do differently is writing more tests and making extra sure that all parts are tested. Some successes we would attempt to recreate include the use and creation of the detailed visual board and the assignment of the different roles.

Key plans were mostly made through Discord, since a majority of the work was done over the break. We attacked this Christmas break problem by planning as much as we could before, so everybody knew what they were doing over the break. The plans were followed to a certain extent but were not always achieved, especially since we fell a bit behind schedule compared to the original time plan. Key decisions were made through team discussions, and we often consulted the team leader for confirmation before finalizing the decisions.

4 Use of Tools

List of tools used:

- GCC (for compilation)
- GitHub and Git (version control on remote and local computers)
- VSCode (code editor and IDE)
- Valgrind (memory checks and leak detection)
- CUnit (unit testing for C programs)
- Clang Format (code style formatting)
- GitHub Actions (continuous integration and automation)
- GProf (function call profiling)
- Time (time profiling for entire program runs)
- LCOV, genhtml (code coverage and HTML report generation)
- Make (build automation tool)
- GDB (for debugging)

We did not lack any tools and we had great use of all of them.

5 Communication, Cooperation and Coordination

Our team primarily communicated through Discord. At the beginning of the project, we held three team meetings to discuss and plan the upcoming weeks. However, during the break, all communication shifted exclusively to Discord. Trying to maintain effective communication, cooperation, and coordination during the break was not easy, as everyone was home for the holidays. We held team meetings around every five days, to review the progress and plan for the upcoming steps.

We have not dealt with a situation where someone was significantly stressed, angry or demotivated in the project. We had designated a role called "Lead well-being officer" at the start of the project. If someone felt stressed, unhappy or demotivated, they could contact this person, but as previously mentioned no one did. We learned that it is very difficult maintaining effective collaboration and communication while not seeing each other in person. Each team member has a responsibility to pull their weight and if some fail to do this, the whole team gets affected.

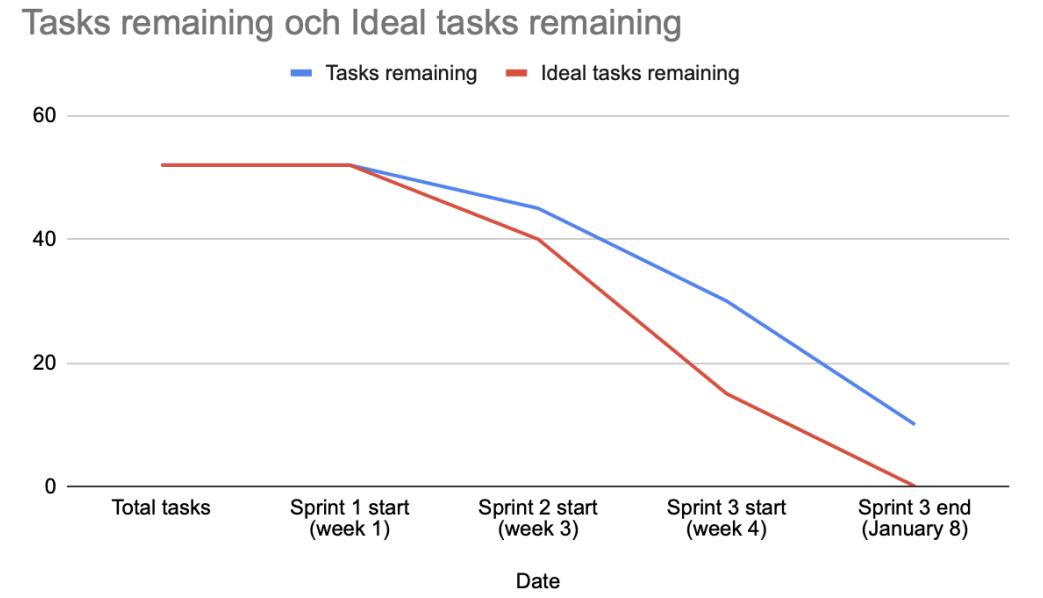
Using a GitHub project for collaboration, each team member worked independently on their assigned issues, but reached out for help when needed. To facilitate collaboration we set up a text channel for pair programming, allowing team members to coordinate and find coding partners for tasks that were larger. During the initial weeks of the project, there was almost no collaboration within the group in the sense of programming in pair. However, in the beginning of January when we entered the compacting phase, teamwork and collaboration improved significantly. This was shown in much more Discord and GitHub communication.

6 Work Breakdown Structure

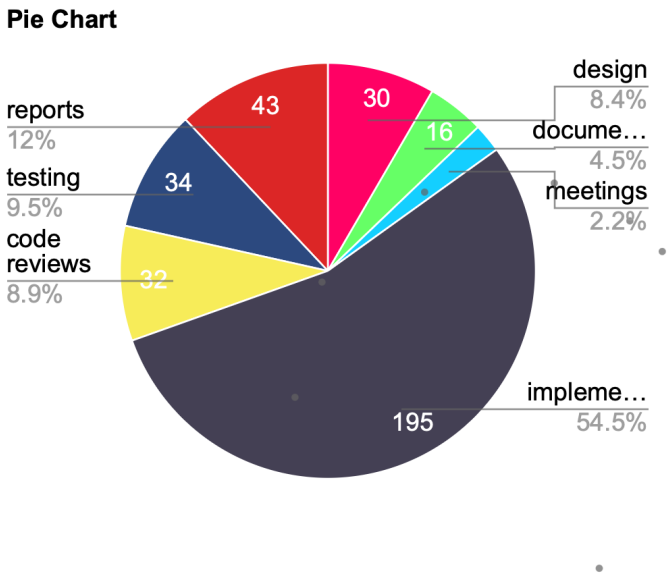
As previously mentioned, we planned for three sprints with each sprint being around one week long. The first sprint was mostly learning and planning while the other two included programming. We did not divide the tasks evenly,

instead each person was in charge of their own work by claiming issues on GitHub. Pair programming was not frequently used throughout the project but became more common towards the compacting and integration phase when the programming tasks became harder.

The tasks were defined as issues in the GitHub project, and were categorized by size (XS, S, M, L) and prioritized as low, medium or high. Our project leader determined the size of each task based on an initial assessment of difficulty. Those tasks identified as XS or S were usually defined as a single function, but for the larger tasks, such as converting a format string to a bit vector, it could be multiple functions. While working on a task, you were expected to create a file containing the function(s), a corresponding header file, and a test file. We tried to load-balance the workload by supporting and helping each other as much as possible. If someone was handling a large task and felt overwhelmed, they could ask the others for help. We also managed the work load by dividing the work as evenly as we could, with some working more on the code, some working on the report and some preparing the presentation.



Figur 2: A burn-down chart with the tasks and sprint plan.



Figur 3: Pie chart of distribution across activities

7 Reflection

We have mostly implemented the right thing, but we did the testing of the entire program a bit too early. As we approached the deadline and prepared for the presentation, we found a large compacting-related bug that we did not come across earlier. The testing was good for most files, but we were too quick to say that absolutely everything works. Upon integration with Assignment 2, it also worked with a minor adjustment, our limiting page size (2040 bytes).

- Process rating

4/7. We are fairly satisfied with our process, but as mentioned earlier there were a few things we could have done differently. We could have had more scrum meetings and better work-load division.

- Delivered product rating

4/7. The program works without any memory leaks, and is sometimes faster than manual memory management. There are however some unimplemented functions, as well as the compacting bug that was discovered at the very end of the project. The code is also a bit hard to understand at certain parts and would need refactoring if we had more time. We could have also added more comments to the code.

- Quality assurance rating

4/7. The product meets the quality standards in a good way, as the code works and does what it is supposed to do without any specific errors. Our code does not allow allocations bigger than our page size (2040 bytes).

4/7. Our biggest win was delivering most of the product before the deadline. After just 1-2 weeks we had fallen behind our time plan and began to worry whether we would finish the product in time. Therefore, we are very pleased that we managed to finish so many functions with some time to spare, before we discovered the bugs. On the other hand, we consider our biggest loss the lack of planning and uneven work load distribution. We should have assigned specific tasks to each person with harder deadlines. The bug that we found at the end of the project was also something that caused us to be a bit disappointed and that decreased the quality of our product. Other than that, we are fairly happy with the finished product.