Uppsala University

Imperativ och objektorienterad programmeringsmetodik

Course code: 1DL221

Conservative Compacting Garbage Collector

Authors:
David Ohanjanian
Edvin Bruce
Fredrik Hammarberg
Olof Lindström
Viktor Kangasniemi
Viktor Wallstén

January 11, 2023

Contents

1 Group name

This group's name is "root".

1.1 Participant List

- **David Ohanjanian**
  E-mail: david.ohanjanian.2221@student.uu.se
  Active dates: November 28, 2022 - January 11, 2023

- **Edvin Bruce**
  E-mail: edvin.bruce.8688@student.uu.se
  Active dates: November 28, 2022 - January 11, 2023

- **Fredrik Hammarberg**
  E-mail: fredrik.hammarberg0804@student.uu.se
  Active dates: November 28, 2022 - January 11, 2023

- **Olof Lindström**
  E-mail: olof.lindstrom.3775@student.uu.se
  Active dates: November 28, 2022 - January 11, 2023

- **Viktor Kangasniemi**
  E-mail: vika6687@student.uu.se
  Active dates: November 28, 2022 - January 11, 2023

- **Viktor Wallsten**
  E-mail: viktor.wallsten.3399@student.uu.se
  Active dates: November 28, 2022 - January 11, 2023

2 Quantification

Project start date: 28/11/2022
Project end date: Hand-in deadline 11/01/2023 at 15.15, presentation 12/01/2023 at 15.15
Number of sprints, their start and end dates: Sprint 1: 05/12/2022-11/12/2022, sprint 2: 12/12/2022-18/12/2022, sprint 3: 19/12/2022-30/12/2022, sprint 4: 02/01/2023-08/01/2023
Total number of new lines of C code written excluding tests and

preexisting code: 1389
Total number of lines of test code: 2329
Total number of lines of "script code" (e.g., make files, Python scripts for generating test data, etc.): 162
Total number of hours worked by the team: estimated to 500 hours in total
Total number of git commits: 177
Total number of pull requests: 1 (see elaboration on this under section 7)
Total number of GitHub issues: 3

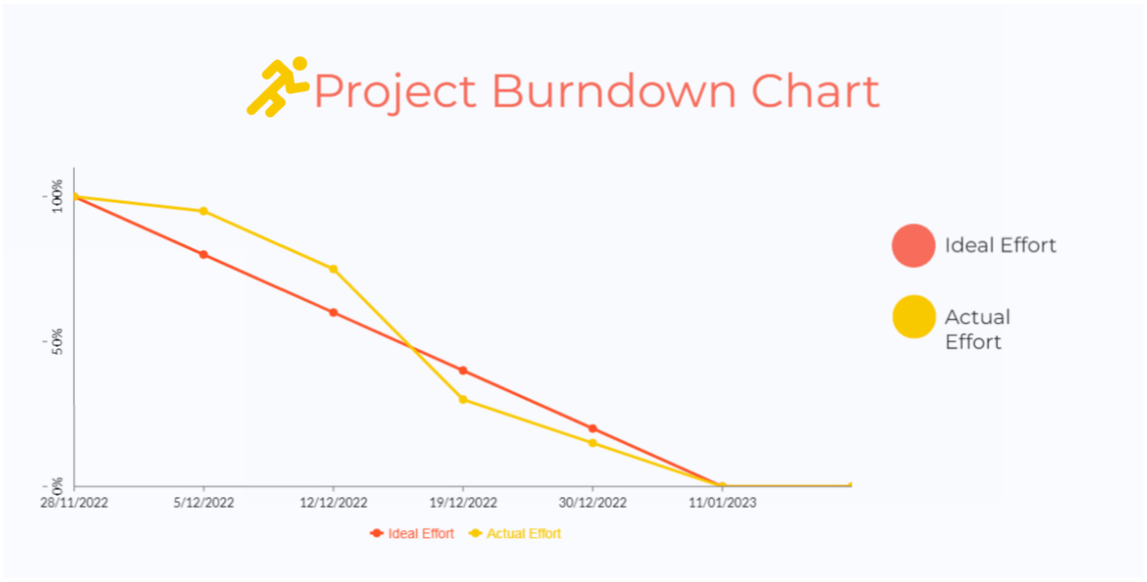

Figure 1: Burndown chart for the project

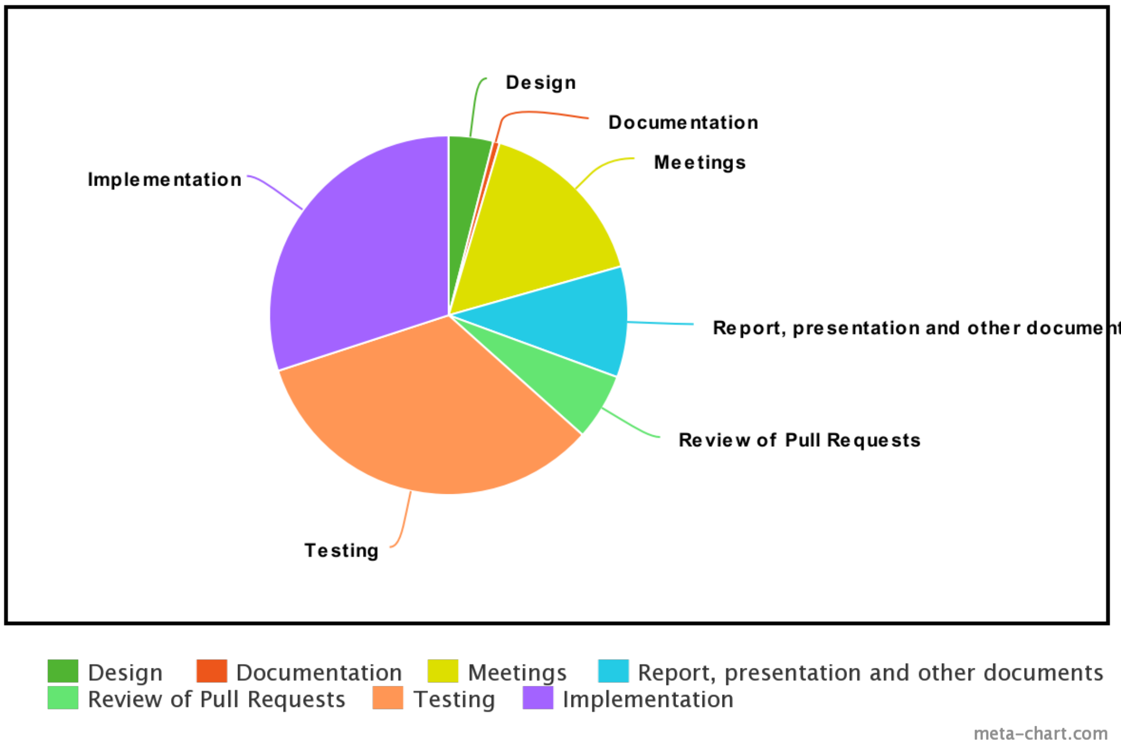

Figure 2: Piechart for estimated time spent on different categories

3 Process

3.1 Inception

During the course of the project we utilized a process loosely based on "Scrum", which was presented to the students in one of the lectures. By taking this approach we divided the project into several smaller sprints. However, in accordance with the Scrum concept, we rejected the old sequential approach of working through the stages of development whereby one sequence has to be fully completed before the next one is being started. Rather than resembling a "relay race", the Scrum process is likened to the collaboration of a rugby team at work. In other words, the sequences are overlapping, i.e., the different stages of development occur simultaneously.[1]

For the sake of sharing the workload we split the team into two groups, both of which were to take on a specific task that was be completed before the end of the current sprint. At a later stage, the team was instead divided into three smaller groups. To illustrate what kind of tasks that were assigned to the groups, an example from the third sprint could be mentioned; namely that of making the, then, somewhat functioning garbage collector compatible with a previously written program, which had been created during the IOOPM course. If more than two people would have tried to implement this simultaneously, the rising risk of conflicting code might well have weighed out the gains of the increase in productivity.

In total, four sprints of varying length were set for the duration of the project. In accordance with our process of choice, after each sprint a meeting would be held, whose objective was to inform the other participants of the state of the project, what progress had been made, and to bring up potential problems that had to be dealt with.

3.2 Implementation

The development of the project aligned itself with the implemented process to a satisfactory degree. We proceeded with the initial list of sprints that were decided upon early on, and we did stick to our plan of having meetings after each sprint. As to how we tackled the break during the Christmas holidays, a two-day hiatus was collectively agreed upon, as well as an additional day off on New Year's Eve. However, the remaining work was to be carried out remotely and independently, although some communication would be carried out using the team's Discord server.

Regarding the realization of the actual product, the process of implementing a stack scanner was separated from the rest of the functionality of the garbage collector such as the heap structure, the designing of the page file, or the collection of metadata. Hence, most of the modules could be worked on at the same time. To make sure that the implementation did not rest on false theoretical assumptions, concurrent testing of the modules took place as the project unfolded. This was necessary in particular of the stack scanner, whose functionality is predicated on the inner workings of the stack, which in turn is partly obscured from view vis-à-vis the programmer.

One of our goals that we were not able to achieve was to make all members participate in the different parts of the project at some point, by rotating through the main areas of development. We set this goal because we considered it a good way to further ensure that every member of the group had some level of insight into each bit of the project, and therefore would have a greater understanding of the project as a whole. The reason as to why this failed was mainly due to the fact that things were progressing at a faster rate than expected, and therefore the reassignment of duties would have drastically brought our momentum to a halt, had we decided to stubbornly stick to this vision.

To elaborate on the drawbacks of our process, the consequences of the initial choice to separate the development of the stack scanner from the rest of the garbage collector has to be scrutinized. Already beforehand we suspected that the work put into the programming of the stack scanner would not equal that of the rest of the functionality, still half of the team would work on its production. This idea was perhaps not a bad one, because we thought that "the sooner it is finished, the sooner new tasks can be delegated". However, it turned out to be less than clear how these two main parts of the product were supposed to fit together. On an abstract level it was obvious, but for the code to actually function the points of intersection had to be utterly precise. This was not as much an issue of communication as it was grounded in a misjudgment of the project at large; it was the apparent trade-off for rejecting the linear approach of development. Instead of quickly moving on to new tasks once the basic structure was set up, a great deal of time was spent on trying to optimize the stack scanner to filter out stack pointers – a feature that turned out to be superfluous and rather insignificant, as the heap structure would provide the interval needed to separate the desired memory addresses from the unnecessary ones. Eventually the stack scanner would amount to a rather concise function, and a number of earlier implementations had been made either redundant, or obsolete. With the benefit of hindsight, it can be concluded that Donald Knuth indeed was correct in famously stating that "premature optimization is the root of all evil (or at least most of it) in programming".[2] Nevertheless, despite the cost of reduced efficiency, the increased insight into the internal structure of the stack did provide a safeguard for false theoretical assumptions that could otherwise have jeopardized the entire project.

4 Use of Tools

During the course of the project we utilized git and GitHub to enable version control and sharing of the project files. We also set up continuous integration for the project repository in order to automatically run tests on pull requests. The tracking of issues through GitHub was employed, although to a lesser extent. We used Trello to better visualize the work that needed do be done, and to help us divide it more evenly between us. Communication was mainly conducted via a Discord server that was set up at the start of the project. Another tool that proved itself to be useful was VSCode's live share feature. This feature allowed one of us to host a coding session which the others could then join using their own computers, in order to conduct parallel coding on the same set of files as the host. This was mainly used during our planned work sessions where all of us attended in person.

5 Communication, Cooperation and Coordination

We did not communicate particularly much with people outside of our group. Only once, at the startup of the project, did we reach out to our group supervisor, for the purpose of delineating the project plan. Occasionally we asked Elias Castegren[1] for advice regarding specific problems that we had encountered, but other than that we did not reach out to anyone outside of the team for help to any extent worth mentioning.

When a group encountered technical difficulties, the first step was always to try and solve it within the group, and if that proved to be too great of a challenge, the entire project group got informed on the issue. If the problem still remained unsolved, we would reach out to external parties for help.

We never had any real issues with members of the group having their contribution to the project hindered by external circumstances.

6 Work Breakdown Structure

As stated previously in this report, we initially divided the team into two groups of three. One group was to work on reading and handling the stack, and the other one was to start working on the rest of the garbage collector. After both tasks were completed, we formed new groups with different team members, this time consisting of three groups of two. Now we had three

[1]Elias Castegren - IOOPM course manager.

different tasks: One group was to work on the presentation, another on writing this report, and the last one was to integrate the garbage collector with an existing program.

Concerning workload balance, the idea was that each member should tackle their task to the best of their abilities. To ensure that no one felt overworked or overloaded it was decided that once a group had finished their task they would help out the other groups if necessary.

The tasks were not always balanced, meaning that each member of the group was not always working on an equally difficult and large problem as every other member. This is not always possible, neither is it desirable, as skill and knowledge differs among people. Rather, the different abilities and skills of the people involved should constitute the sole basis of the cooperation. The unique dynamics adhering to a team requires of each of its members that they find their own niche. This is sometimes harder and sometimes easier. As these roles will inevitably vary depending on the composition of the group, the focus must remain on the path to realizing the project, and how to overcome the obstacles in order to get there. Nevertheless, for a functioning teamwork, each member is required to contribute.

7 Quality Assurance

7.1 Tests

We are quite confident in the fact that our garbage collector works; this it due to our rigorous testing, with over 2000 lines of test code in 66 individual tests. In addition to this, the fact that we successfully managed to implement our garbage collector in a previously written program that is quite extensive, makes us confident that we have written well-functioning code. We of course do not claim that our tests are 100% exhaustive, but we do not see any reason to feel unsure about the performance of the delivered product.

Test were written after the completion of each module, e.g., the module containing the page struct and its corresponding functions, or the filtering of memory addresses in the stack scanner. Every member of the group took part in writing tests at different stages of development.

We utilized unit-, integration- and regression testing of our program. Unit testing was utilized to ensure functionality on singular functions, which makes bugs easier to find. Every time new code was pushed to our GitHub-repository, regression tests were automatically run, as set up by continuous integration, to assure that no faulty code was being employed. Lastly, the process of integrating the garbage collector with an interactive program concluded the testing phase.

7.2 Pull Requests and the Reviewing of Code

At the start of the project, we agreed on the importance of properly using branches during the entire development. The idea was to start a new branch for each major module, and later merge these branches with the master branch of the repository once they were finished. This was to be carried out by means of GitHub's pull request feature. All team members had to be present during the merging of a pull request for the purpose of giving their approval of the code. Developing using multiple branches seemed like a more professional way of conducting a project, rather than simply pushing all code to the master branch, as it acts as a safety net, should something go wrong with the code or its implementation. Furthermore, the employment of continuous integration guaranteed that all tests were passed before a branch could be merged with the master branch of the repository.

Although these ideas were sound, the reality turned out to be different. The two initial branches – one for the "start of the GC" and another titled "stack crawler sandbox" – unintentionally became the development branches belonging to the respective groups. As the process of refactoring and finishing the stack scanner dragged on, drafts of the other modules were started by the other group and simply pushed to their branch. As this branch ended up containing most of the files, the final version of the stack scanner was simply imported to that branch as well. This was in a way reasonable, because the making of a pull request at that point felt superfluous. The main deviation from the plan probably occurred when new branches were not created, a fault that we attribute to the fact that we were all too immersed in the process at that time. However, a final pull request was made once all the modules were completed and gathered in the same branch, during which the entire team was present. Although we do consider the lack of pull requests a shortcoming of the project, we at least remained loyal to this central principle of software development, as well as making use of additional branches in order to steer clear of the master branch as the only place for commits.

Concerning the reviewing of code, this was not only done during the one and final pull request, but something that continuously took place throughout the entire project. Every sprint meeting started out with a collective examination and presentation of the current state of the code. This was useful for providing the other team members insight into the overarching evolution of the project. As we partly programmed together during our programming sessions, which became extensions to the meetings, the reviewing of code also occurred as a natural way of cooperation, as we tried to solve certain problems collectively. To state a few concrete examples of where this was beneficial, the following could be mentioned:

- It was decided that the struct of the stack scanner should make use of double void pointers for storing its elements and filtered addresses.
- The stack scanner was later on to be refactored not to contain a struct, but rather to be implemented as a module containing a single function returning a triple void pointer.
- Certain ways of filtering the addresses of the stack scanner were discussed and as a result abandoned.

The preferred way of tracking issues on GitHub was unfortunately not used to a particularly large degree. One reason for this might be that the frequency of our meetings was relatively high, and that the direct communication pertaining to meetings attended in person were deemed more appropriate for the problem solving of the issues at hand. Another reason may be the perceived novelty of this particular tool, with which all members were not familiar due to not yet having completed the code review achievements of the course. Nevertheless, a total of three GitHub issues were opened and subsequently closed after completion.

7.3 Formatting and Style

To guarantee a consistent formatting style and a comfortable reading experience, all the code of the project was formatted using the GNU C coding standard.[3]

8 Reflection

(Ratings from one to seven)

Satisfaction with process: 7
We are all united in the belief that we had a well thought-out development procedure throughout the project. Although the choice of splitting up into two groups at an early stage of the project proved to have some disadvantages, we certainly would have seen a drastic decrease in productivity had we not done so. This was also a way of ensuring that all members took part in the development.

Satisfaction with delivered product: 7
We are very satisfied with the finished product, as we have been able to integrate our garbage collector with a previously written interactive program without any memory leaks. In addition to this, we have also been able to optimize the program quite a bit, which is now running five times faster than the original version.

Satisfaction with quality assurance: 7
We are satisfied with the quality assurance of our finished product. This is the case as previously motivated. We are happy about the amount of tests we have written, and the fact that integration between our finished product and a previously written program works splendidly gives us great confidence regarding the quality of our product.

The team considers our greatest accomplishment to be the delivering of a well written program in conjunction with the utilization of the Scrum-based methodology, which we held on to from the beginning of the project to its completion. We are very happy with the experience of the workflow, how we manged to stick to the time frame that we had set out, and the way in which we overcame the various obstacles that could potentially threaten or cripple the development of a project.

The team considers our biggest failure to be the fact that we did not work with pull requests as we had intended to. As we separated into working in two groups, we did not make any pull requests until the very end of the project. Furthermore, we did not utilize GitHub in the way we wanted, as we did not fully develop the routine of formally reviewing code by tracking issues, but rather examined the code together during meetings and programming sessions. In addition to this, the premature optimization of the stack scanner taught us an important lesson on how to prioritize during a programming project.

References

[1] H. Takeuchi and I. Nonaka, “The new new product development game,” *Harvard Business Review*, 1986.

[2] D. E. Knuth, “Computer programming as an art,” *Commun. ACM*, vol. 17, p. 667–673, dec 1974.

[3] R. Stallman et al., “GNU coding standards.” https://www.gnu.org/prep/standards/standards.pdf, 2021. Accessed: 2023-01-06.