A collaborative peer review process for grading coding assignments

Pratik Nayak, Fritz Göbel, and Hartwig Anzt

Steinbuch Center for Computing, Karlsruhe Institute of Technology, Germany.

Abstract. With software technology becoming one of the most important aspects of computational science, it is imperative that we train students in the use of software development tools and teach them to adhere to sustainable software development workflows. In this paper, we showcase how we employ a collaborative peer review workflow for the homework assignments of our course on Numerical Linear Algebra for High Performance Computing (HPC). In the workflow we employ, the students are required to operate with the git version control system, perform code reviews, realize unit tests, and plug into a continuous integration system. From the students' performance and feedback, we are optimistic that this workflow encourages the acceptance and usage of software development tools in academic software development.

Keywords: Peer review · Continuous Integration · Collaborative learning · Sustainable software development.

1 Introduction

With the digital revolution, much of the research, engineering, and production is no longer realized by the human workforce but automatized and controlled by computer programs. This radically changes the labor market and the skill-set wanted by employers. While a significant portion of the automate-able work will be handled by robots and computer programs in the future, other skills such as expertise in developing software and using the tools that enable sustainable software development will be important. Though there exist tutorials and talks discussing the use of development tools, experience tells us that only the practical use of the tools prepares the researchers for operating them in larger software projects. Against this background, we decided to expose graduate students to the practical use of sustainable software development tools by establishing a collaborative peer review concept for grading coding assignments.

In this paper, we elaborate on how we encourage the use of sustainable software development paradigms and enforce the usage of software development tools in a course on *Numerical Linear Algebra for High Performance Computing* offered at the Karlsruhe Institute of Technology. The course content includes the design of efficient algorithms for basic linear algebra operations, direct and iterative linear solvers, hierarchical methods, preconditioners, etc., concentrating on massively parallel architectures such as multi-core CPUs and GPUs. By the

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77980-1_49

end of the course, we expect the typical student to be able to use the techniques of parallel programming they have learned in the course and apply them to their thesis projects and their future coding endeavours. Additionally, we want to provide them with some knowledge about paradigms for sustainable software development and give them hands-on experience in using tools that are popular in the realization of software projects. We reiterate that the course does not focus on programming tools and software engineering paradigms but on the design of high performance computing algorithms, and the course content can be taught without touching the topic of sustainable software development. But we use our first-hand experience in large software efforts to propagate software sustainability paradigms and require the use of collaborative software development tools in the homework assignments.

Before detailing in Section 3 the peer review workflow we employ for the homework assignments, we provide in Section 2 some background about the tools we use for this. In Section 4 we discuss our experience with the approach, and conclude in Section 5.

2 Background

For sustainable software development, a set of tools has proven to ease software development and maintenance. While there exist tutorials, textbooks, and seminars on the use of those tools, we want to review some of them that we consider most important and thus include in our homework peer review workflow.

Version control systems (VCS). Version control systems are a popular way to manage codebases. They started as a snapshot capability, enabling the developer to roll-back the code history, thereby easing debugging and maintenance. They evolved as powerful platforms for collaborative software development. Some of the popular open-source version control systems are: Revision Control System (RCS), Concurrent Version Control (CVS), Subversion (SVN), Mercurial, and Git.

Figure 1 shows a typical version control workflow with the git version control system. In this workflow, a developer can create a feature branch from the main branch, add new functionality or change existing functionality, and create a merge request (MR) to merge the feature branch back to the main branch. The process where peers inspect and criticize the code changes and provide feedback to the developer is called a *code review*. This step may appear tedious, but is one of the most important components of sustainable software development. It is essential that both the feedback-giving peer and the developer take the code review seriously, and the developer acknowledges and adopts the reviewers' comments. Once all flaws and improvement suggestions are taken care of, the reviewers approve the merge request and the changes are merged into the main branch. The git version control system orchestrates the merge in case multiple developers want to merge to the main branch and enables the developers to re-

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77980-1_49

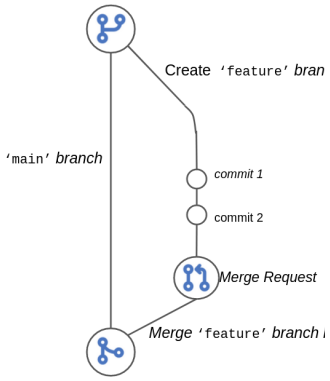

Fig. 1: A typical git workflow. [1]

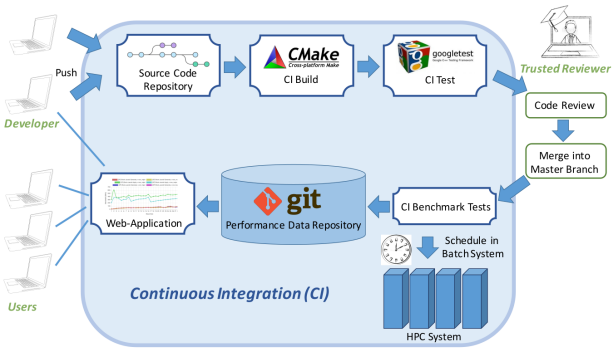

Fig. 2: A software development workflow [8]

trieve new changes from the main branch into the feature branch.

Continuous integration. With the increasing complexity of the coding tool-chain, from different hardware platforms to the different build systems and the different compilers that need to be supported, it becomes increasingly challenging to ensure code robustness. Continuous integration (CI) aims to improve the reliability of a codebase [2, 3, 5] by automatically checking the code correctness after a change to the codebase has been pushed through the version control system. Specifically, the CI replicates the complete code usage process from source code cloning to the compilation of the code in a specified hardware-software environment. This is important in particular if the software is developed as a collaborative effort, and distinct developers are independently making changes to the codebase. The goal of CI is to ensure the permanent compileability of the codebase on the target system and to notify the developers about compilation issues. In addition to verifying whether the codebase compiles, CI systems usually also employ an automated testing workflow.

Automated Testing. Automated software testing pursues the idea of having an automatic mechanism that verifies the correctness of all functionality of a software ecosystem. Of course, this is a very complex goal, as there exists a large number of use cases and functionality combinations. Ultimately, testing all combinations, and all end-to-end user applications would be infeasible in terms of effort and time. Therefore, one typically uses a hierarchical testing strategy: On the lowest level, each basic functionality is accompanied by a *unit test*. This function-specific unit test verifies the correctness of the basic routine for all possible input and output scenarios. A failing unit test can easily point the developer to the part of the code that needs debugging.

As the correctness of all basic functionality (passing unit tests) does not guarantee the correctness of functionality combinations, unit tests are usually

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77980-1_49

complemented by *integration tests* that form the second level of the testing hierarchy. These tests verify the correctness of different functionality combinations. Often, it is impossible to cover all combinations due to the sheer quantity of the possible permutations. Therefore, integration tests usually only cover the most popular functionality combinations.

Finally, the third level of automated testing is formed by *end-to-end tests*, which try to emulate a user's workflow. As it is again impossible to emulate all possibilities, end-to-end tests only emulate a few, typical use cases. For all levels of automated testing, there exist sophisticated tools such as GOOGLETEST [7] and the CATCH2 TEST FRAMEWORK [6] that provide frameworks.

Services in the Cloud. Gitlab and Github offer platforms that allow users to run Continuous Integration pipelines and automated tests on remote servers in the cloud. They also offer web interfaces for interactive and collaborative coding using ideas such as Merge Requests (MRs). Using these services removes the need to procure and run servers that provide version control systems, continuous integration, and automated testing.

3 Methodology

Numerical linear algebra for High Performance Computing (HPC). The course we offer at KIT is aimed at Masters students seeking to learn about computational numerical linear algebra methods suitable for computational science and its realization in HPC settings. To encourage the students to apply these techniques in larger projects, we require a final course project instead of an end-of-term examination. We offer some project ideas, particularly to enable them to contribute to GINKGO [9], but also encourage the students to come up with ideas or extend their thesis works. Additionally, the students have to complete several homework assignments where they are required to develop a parallel version of an algorithm, implement and run benchmarks on an HPC architecture, and analyze the algorithm performance in a report.

To prepare the students for collaborative coding, an exercises framework is provided in a version control system which handles the automatic compilation and testing of the code written by the students [10]. This allows the students to focus on the algorithms and parallel programming ideas rather than spending time on the build system. An additional advantage is the uniformization of the build and execution process for all students making it easier to debug and help the students.

The Peer review process. Figure 3 shows a schematic of this peer review process. Each student creates a fork of the exercise framework and a student-labeled sub-folder from the main folder in which they add their implementations. On the submission date, the CI is run (which compiles and runs their code with the unit tests) and the students create an MR for their exercise. The MRs are assigned to their peers in a round-robin fashion, and with the help of the

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77980-1_49

Gitlab interface, they can provide feedback to their assigned peer. They are encouraged to follow code reviewing guidelines [4] while reviewing their peers' code. The review process is iterative, and the students are encouraged to update and enhance their code according to the feedback received. After approval by the reviewing peers, the students' codes are merged into the main repository to replicate an actual project workflow.

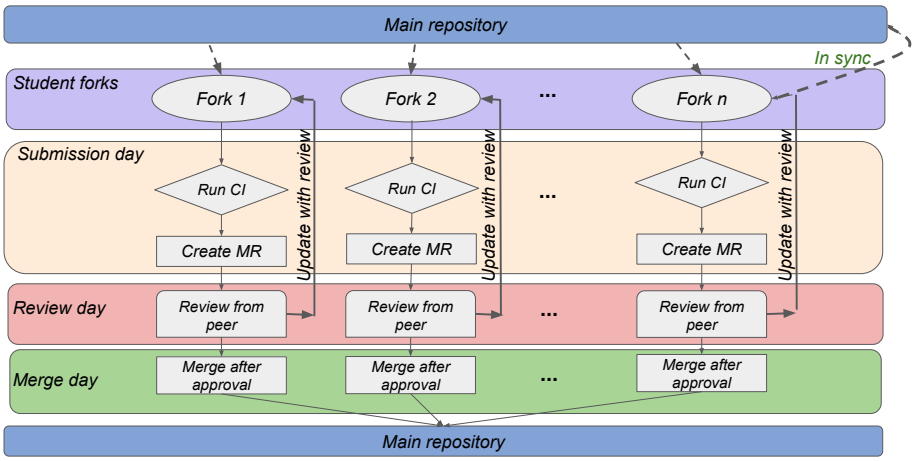

Fig. 3: The peer review process

Figure 4 shows a snippet of a review by Student 1 of the code written by Student 2. Student 1 is assigned to the MR opened by Student 2 and can provide comments on specific lines of the code based on a diff (the difference between the code in the target branch against the code added by the student).

Grading. In addition to the coding assignments, the students are required to work on a final course project. We split the final grade into three parts: 40% of the grade is made up by the project code and report, 30% is assigned for a short (approximately 10 minutes) presentation on the student's project, given at the end of the term, and the remaining 30% of the grade is made up by the homework assignments. (70% of the points are required to pass the course.) Each homework assignment carries a maximum of 10 points. A performance analysis report makes up 4 of the 10 achievable points. Another 4 points are assigned for the code, where 2 points are awarded for functioning code, 1.5 points for the code quality, 2 points for the quality of the review a student provides, and 0.5 points for incorporating the review feedback that they receive from their peers.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77980-1_49

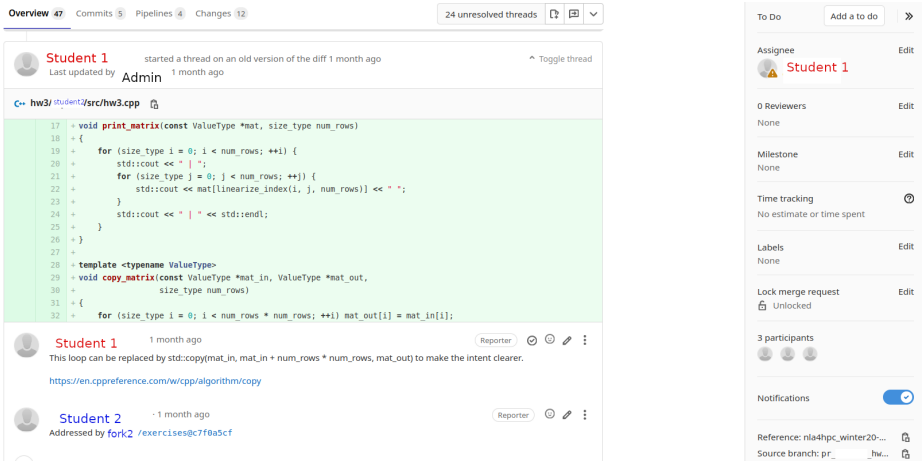

Fig. 4: A typical peer review comment

4 Discussion

The motivation for establishing this peer review process for the homework assignments is that we are convinced that the practical experience in using software development tools, CI, and adhering to a peer review process is an essential skill for graduates entering the software-focused labor market.

In addition to equipping students with practical experience in using tools for sustainable software development, we observed that the framework allowed students to concentrate on the algorithms and their implementations rather than concerning themselves with tackling the issues from the build systems. We also observed a significant improvement in the code quality of most students throughout the course with students acknowledging the feedback received in the peer review process in the concurrent exercises, which has not been the case in previous versions of the course.

Table 1 shows the feedback we received from the students on the homework workflow. The questions are rated from 1 to 5 with 1 being the best rating meaning that it was very useful/very easy and 5 being the worst rating meaning that it was not useful at all/very difficult.

5 Conclusion

With the wide and easy access to computing and the rise of Open-Source software, it is necessary that we train our students to be familiar with tools for sustainable software development. In this paper, we elaborate on how we introduced a collaborative peer review workflow for the homework assignments in a course on high performance computing. In the future, we plan to enhance the framework with automatic benchmarking of the coding assignments and test the workflow's viability for other courses.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77980-1_49

Table 1: Student feedback

Question	**Avg Rating (1–5)**
How easy was it to use the framework ?	2
How useful did you find the exercises instructions ?	2
How easy was it to compile and run the code as provided ?	2.3
How useful was the code review from your peer ?	1.6
How easy was the reviewing process ?	3.6
Would you like to see this type of frameworks in other courses ?	1

6 Acknowledgements

The authors were supported by the "Impuls und Vernetzungsfond of the Helmholtz Association" under grant VH-NG-1241. The authors would also like to thank Jan-Patrick Lehr of TU Darmstadt for his helpful discussions and perspectives on this subject.

References

1. Git cheat sheets. https://training.github.com/
2. GitLab CI/CD. https://docs.gitlab.com/ee/ci/
3. Jenkins CI. https://www.jenkins.io/index.html
4. The standard of code review. https://google.github.io/eng-practices/review/reviewer/standard.html
5. Travis CI - test and deploy your code with confidence. https://travis-ci.org/
6. Catchorg/Catch2. Catch Org (Jan 2021)
7. Google/googletest. Google (Jan 2021)
8. Anzt, H., Chen, Y.C., Cojean, T., Dongarra, J., Flegar, G., Nayak, P., Quintana-Ortí, E.S., Tsai, Y.M., Wang, W.: Towards continuous benchmarking: An automated performance evaluation framework for high performance software. In: Proceedings of the Platform for Advanced Scientific Computing Conference. pp. 1–11. PASC '19, Association for Computing Machinery, New York, NY, USA (Jun 2019). https://doi.org/10.1145/3324989.3325719
9. Anzt, H., Cojean, T., Chen, Y.C., Flegar, G., Göbel, F., Grützmacher, T., Nayak, P., Ribizel, T., Tsai, Y.H.: Ginkgo: A high performance numerical linear algebra library. Journal of Open Source Software (Aug 2020). https://doi.org/10.21105/joss.02260
10. Nayak, P.: Pratikvn/nla4hpc-exercises-framework (Jan 2021)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77980-1_49