Peer Code Review for Scientific Software

A Postion Paper for the 2020 Collegeville Workshop on Scientific Software

Jeffrey C. Carver and Nasir U. Eisty
Department of Computer Science
University of Alabama
carver@cs.ua.edu
neisty@crimson.ua.edu

1 Background

Historically, developers of business/IT software have employed various types of testing techniques to improve the quality of their software. However, developers of research software have found it more difficult to employ some of the traditional software testing techniques [3]. The complexity of the underlying research domains leads to research software that has complex computational behavior. This complexity, along with the fact that the expected outputs are often unknown, make it difficult to define appropriate tests and identify input domain boundaries [5]. In many cases, the input space of research software is so vast that it is not feasible, or even possible, for a developer to create a test suite that adequately exercises the limits of the software [7]. Therefore, while testing is useful, it is generally not sufficient to ensure the quality of research software.

Conversely, peer code review is a lightweight, asynchronous method for ensuring high-quality code [1]. Peer code review is a systematic examination of source code by peers of the software's developer to identify problems the developer can then address. Recently, commercial organizations and open source projects have been adopting peer code review as a more efficient, lightweight version of the older, more formal inspection process [6]. While peer code review is effective and prevalent in open-source and commercial software projects, it remains underutilized in research software.

By employing peer code review in their projects, research software developers will see benefits both in the short term, through higher quality scientific results produced by high-quality software, and in the long term, through creation of more maintainable software [4]. The higher quality scientific results occur because developers focus their attention on the code itself to identify mistakes, inefficiencies, and other aspects of the code that need improvement. The improved maintainability arises because as team members start reviewing each others's code, they begin writing more readable code to enable the peer-review process. Code that is more readable and easier to understand is also more maintainable over time.

In addition to improving general software quality, the use of peer code review has other specific benefits in the research software domain. Unlike traditional commercial/IT software, research software developers are often exploring new scientific or engineering results, which may be unknown *a priori*. The lack of an oracle makes it difficult for developers to create adequate tests that can diagnose whether a result is a new insight from a simulation or is the consequence of a fault in the software [2]. Even in cases where the expected output is known, the complexity of the software often makes it impossible to adequately test all important configurations of the software and input data. Conversely, when a person conducts a code review, he or she is able to analyze the underlying

algorithm and identify problematic conditions. Therefore, while peer code review is essential for any type of software, it is even more important for research software.

2 Tutorial

To address the need for more peer code review in scientific software we have developed a tutorial. The focus of the tutorial is to provide an overview and motivation for the use of peer code review in scientific and research software development. The tutorial covers topics including:

- Why perform peer code review
- Goals of peer code review
- Code review practices both for the reviewer and for the developer
- Code review techniques
- How to provide good feedback

We have delivered this tutorial in a number of conference and national lab venues, including during the 2019 Exascale Computing Project Annual Meeting `https://se4science.org/tutorials/ECP19/`. The ideal use of such a tutorial is to provide an overview of the general practices of code review, then to work more directly with individual teams to identify specific items of interest in their domain. In addition to advocating for peer code review in general, we are interested in delivering the tutorial in other venues.

References

[1] A. F. Ackerman, L. S. Buchwald, and F. H. Lewski. Software inspections: an effective verification process. *IEEE Software*, 6(3):31–36, May 1989.

[2] D. Heaton and J. C. Carver. Claims about the use of software engineering practices in science: A systematic literature review. *Information and Software Technology*, 67:207 – 219, 2015.

[3] U. Kanewala and J. M. Bieman. Testing scientific software: A systematic literature review. *Information and Software Technology*, 56(10):1219 – 1232, 2014.

[4] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 192–201, New York, NY, USA, 2014. Association for Computing Machinery.

[5] H. Remmel, B. Paech, P. Bastian, and C. Engwer. System testing a scientific framework using a regression-test environment. *Computing in Science Engineering*, 14(2):38–45, 2012.

[6] P. Rigby, B. Cleary, F. Painchaud, M. Storey, and D. German. Contemporary peer review in action: Lessons from open source development. *IEEE Software*, 29(6):56–61, Nov 2012.

[7] S. A. Vilkomir, W. T. Swain, J. H. Poore, and K. T. Clarno. Modeling input space for testing scientific computational software: A case study. In M. Bubak, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, editors, *Computational Science – ICCS 2008*, pages 291–300, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.