Testing Scientific Software

A Postion Paper for the 2020 Collegeville Workshop on Scientific Software

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

1 Background

Testing is a useful practice for producing high-quality software. Unfortunately, because of its complex computational behavior, it is very difficult to test scientific software. Scientific software developers often build software based upon a set of mathematical equations and use mathematical analysis to verify the corrections of the computational model [10, 15]. For example, scientists use scientific software to determine the impact of modifications to nuclear weapon simulations since real-world testing is too dangerous and not allowed [15].

While testing is a useful practice, there are some technical challenges for testing scientific software. The first challenge is the lack of test oracles [8]. An oracle is pragmatically unattainable in most of the cases for scientific software because scientists develop software to find previously unknown answers. Due to the lack of test oracles, scientific software developers often use judgment and experience to check the correctness of the software. The second challenge is the large number of tests required to test scientific software using standard testing techniques. Also, the large number of input parameters makes it challenging to manually selecting a sufficient test suite [17]. Finally, the presence of legacy code makes testing scientific software very challenging [1].

A previous systematic literature reported testing challenges due to the characteristics of scientific software and the cultural differences between scientificers and more traditional software engineers [9]. The authors subdivided the testing challenges resulting from the characteristics of scientific software into four categories: a) test case development, b) producing expected test case output values, c) test execution, and d) test results interpretation. Then subdivided the testing challenges resulting from the cultural differences between scientific software developers and more traditional software engineers into three categories: a) limited understanding of testing concepts, b) limited understanding of the testing process, and c) not applying known testing methods.

Because of these challenges, scientific software developers are unlikely to use systematic testing to check the correctness of their code [8, 13, 11]. Even though these developers conduct validation checks to ensure the software correctly models the physical phenomenon of interest [10, 13], there is still a need for testing that identifies differences between the model and the code [5]. In addition, sometimes the reason for limited use of systematic testing results from the testing challenges posed by the software itself [2].

It is not clear how scientific software developers actually perform testing. Scientific software developers commonly omit Unit testing because they have misconceptions about the benefits and difficulties of implementing unit tests [1]. Scientific software developers under-utilize verification testing because they are unaware of the need for it and the methods for applying it [3]. In many cases, scientific software projects do not even include automated acceptance testing and regression testing [14]. Furthermore, the scientific software community is lagging behind in the use of available testing tools, at least partially due to the wide use of FORTRAN [16, 3, 12].

Moreover, there is a lack of recognition for the skills and knowledge required for software development in scientific software development [7]. These developers are typically unfamiliar with available testing methods [4, 6]. As a consequence, they do not usually have a set of written quality goals. Scientists even treat software development as a secondary activity. Because of all these factors, there is a need of proper training on software testing to motivate scientific software developers and perform testing activities in practice.

2 Tutorial

While many testing techniques are beneficial in business/IT software, these techniques are under-utilized in scientific software. To remedy this situation, we are developing a hands-on tutorial entitled "Automatic Testing in Scientific Software" supported by a BSSw fellowship award. The tutorial will start with background information and about the usefulness of using automatic testing techniques in scientific software development. During this portion of the tutorial, we will present challenges, potential solutions, and unsolved problems faced while testing scientific software from our recent survey on testing research software. The second part of the tutorial will be hands-on. We will cover different testing techniques, such as input space partitioning, test-driven development along with some testing techniques specially designed for scientific software such as metamorphic testing and run-time assertion. We will conclude the tutorial with a large group discussion to gather input from the participants about which approaches are more suitable for their individual projects. We will submit tutorial proposals to different conferences and national labs. We would like to present a BSSw webinar once the tutorial is more mature. We are interested in delivering the tutorial in other venues as well.

References

[1] T. Clune and R. Rood. Software testing and verification in climate model development. *IEEE Software*, 28(6):49–55, Nov 2011.

[2] S. M. Easterbrook. Climate change: A grand software challenge. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, pages 99–104. ACM, 2010.

[3] S. M. Easterbrook and T. C. Johns. Engineering the software for understanding climate change. *Computing in Science Engineering*, 11(6):65–74, Nov 2009.

[4] S. L. Eddins. Automated software testing for matlab. *Computing in Science Engineering*, 11(6):48–55, Nov 2009.

[5] P. E. Farrell, M. D. Piggott, G. J. Gorman, D. A. Ham, C. R. Wilson, and T. M. Bond. Automated continuous verification for numerical simulation. *Geoscientific Model Development*, 4(2):435–449, 2011.

[6] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson. How do scientists develop and use scientific software? In *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 1–8, May 2009.

[7] C. Hill. . In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–1, Sep. 2016.

[8] D. Hook and D. Kelly. Testing for trustworthiness in scientific software. In *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 59–64, May 2009.

[9] U. Kanewala and J. M. Bieman. Testing scientific software: A systematic literature review. *Inf. Softw. Technol.*, 56(10):1219–1232, Oct. 2014.

[10] D. Kelly, D. Hook, and R. Sanders. Five recommended practices for computational scientists who write software. *Computing in Science Engineering*, 11(5):48–53, Sep. 2009.

[11] D. Kelly, R. S, R. Saint, P. Floor, R. Sanders, and D. Kelly. The challenge of testing scientific software. In *in Proc. Conf. for the Association for Software Testing*, pages 30–36, 2008.

[12] G. S. Lemos and E. Martins. Specification-guided golden run for analysis of robustness testing results. In *2012 IEEE Sixth International Conference on Software Security and Reliability*, pages 157–166, June 2012.

[13] C. Murphy, M. Raunak, A. King, S. Chen, C. Imbraino, G. Kaiser, I. Lee, O. Sokolsky, L. Clarke, and L. Osterweil. On effective testing of health care simulation software. *Technical Reports (CIS)*, 05 2011.

[14] L. Nguyen-Hoan, S. Flint, and R. Sankaranarayana. A survey of scientific software development. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 12:1–12:10. ACM, 2010.

[15] D. E. Post and R. P. Kendall. Software project management and quality engineering practices for complex, coupled multiphysics, massively parallel computational simulations: Lessons learned from asci. *The International Journal of High Performance Computing Applications*, 18(4):399–416, 2004.

[16] R. Sanders and D. Kelly. Dealing with risk in scientific software development. *IEEE Software*, 25(4):21–28, July 2008.

[17] 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. Springer Berlin Heidelberg, 2008.