SAND2011-1158C

Improving CSE Software through Reproducibility Requirements POSITION PAPER

Michael A. Heroux*
Sandia National Laboratories
Albuquerque, NM
maherou@sandia.gov

ABSTRACT

It is often observed that software engineering (SE) processes and practices for computational science and engineering (CSE) lag behind other SE areas [7]. This issue has been a concern for funding agencies since new research increasingly relies upon and produces computational tools. At the same time, CSE research organizations find it difficult to prescribe formal SE practices for funded projects. Theoretical and experimental science rely heavily on independent verification of results as part of the scientific process and computational science should have the same regard for independent verification but it does not.

In this paper, we present an argument for using reproducibility and independent verification requirements as a driver to improve SE processes and practices. We describe existing efforts that support our argument, how these requirements can impact SE, challenges we face, and new opportunities for using reproducibility requirements as a driver for higher quality CSE software.

Categories and Subject Descriptors

D.2.9 [**Software**]: Software EngineeringManagement[Software quality assurance]

General Terms

Reproducibility,Verification,Software Engineering

Keywords

Computational Science and Engineering, Scientific Process, Software Engineering

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of EnergyÕs National Nuclear Security Administration under contract DE-AC04-94AL85000.

Third International Workshop on Software Engineering for Computational Science and Engineering Honolulu, Hawaii USA

1. INTRODUCTION

Computational Science and Engineering (CSE) is often called the third paradigm of scientific discovery. It is truly a scientific endeavor attempting to add to the global collection of human knowledge. At the same time, as is often observed, CSE software engineering practices lag behind other areas. In particular, research-oriented software is often discarded after the project that produced it is complete. This issue has been a concern for funding agencies such as the US National Science Foundation (NSF) (see the Software Infrastructure for Sustained Innovation–SI^2 for the NSF effort to address this issue [16]), since NSF and similar agencies find that new research increasingly relies upon and produces computational tools. NSF-funded research can typically produce very good research results, but often the software generated as part of the project has been developed only to produce research results, after which the software is set aside, perhaps to be used later, but certainly not available to external people and often not under version control.

CSE research organizations find it difficult to prescribe formal software engineering practices for funded projects because there is not a lot of specific literature on SE for CSE research software, and because researchers would find such prescriptions artificial and a hindrance to progress on their research.

Theoretical and experimental science rely heavily on independent confirmation of results as part of the scientific process. In principle, computational science should have the same regard for independent confirmation, but a study of the literature shows that in many cases there is very little hope of reproducing published results from the published information. Furthermore, results are often generated by software that is not managed and would be impossible to confidently assert as the software that generated the results.

In this paper we discuss how requiring independent confirmation of CSE results can impact the quality of research software. Furthermore, we observe that, although funding agencies may find it difficult to impose formal software engineering practices on funded project, it is very reasonable to expect research teams to obtain independent verification of their results. In fact, it is our opinion that if independent verification is expected, we will see rapid adoption and adaptation of software engineering practices in the research community, not because formal process were imposed but because researchers will want to minimize the overall effort that goes into achieving independent verification. Good software engineering is a part of the process.

2. REPRODUCIBILITY AND SE

If a research team is told it must be able to readily reproduce computational results and support independent results verification, several things become immediately useful:

1. Source management tools: In order to guarantee that results can be reproduced, the software must be preserved so that the exact version used to produce results is available at a later date.

2. Use of other standard tools and platforms: In order to reduce the complexity of an environment, standard software libraries and computing environments can be used.

3. Documentation: Independent verification requires that someone else understand how to use your software.

4. Source code standards: Improves the ability of others to read your source code.

5. Testing: Investment in greater testing makes sense because the software will be used by others.

6. High-quality software engineering environment: If a research team is serious about producing high-quality, reproducible and verifiable results, it will want to invest in a high-quality SE environment to improve team efficiency.

Requiring reproducibility and independent verification does have many positive influences on SE practices and processes. In fact, we believe that reproducibility and verification requirements are the most natural way help CSE teams make progress against SE goals.

3. EXISTING EFFORTS

There are many ways to make progress on software quality and independent verification. In this section we discuss several efforts that have made an impact on research software. This is not an exhaustive list, but is meant to illustrate the variety of ways we can attack this problem.

3.1 ASCI SQA Requirements

In the late 1990s, the US Department of Energy established the Accelerated Strategic Computing Initiative (ASCI, now ASC) [8]. A major component of the program was the establishment of software quality assurance (SQA) policies and a software quality engineering (SQE) team. Any software research effort funded by the program was expected to adhere to the SQA policies and show how it was performing SQE. Assessments and audits were performed regularly. The presence of SQA/SQE in ASCI had a profound impact on software planning and team organization. One of the biggest impacts was the creation of the Trilinos Project [13, 14] at Sandia National Laboratories. Although mathematical research software had been developed at Sandia for many years by small teams, each with its own custom software stack and informal processes, the rigors of SQA/SQE in ASCI made it infeasible for small ad hoc teams to satisfy requirements independently. Trilinos was started in part to provide SQA/SQE support for small teams. Once started, the Trilinos Project continued its focus on SQA/SQE issues, above and beyond what ASCI required, primarily because Trilinos development teams saw the value of improved software process and tools and the cost of improvement was leveraged across the many teams that make up the project. Presently Trilinos supports nearly 60 small team package development efforts, providing high quality software processes and tools that allow the large collection of independently developed, inter-dependent software packages to work in concert.

3.2 The ACM Transactions on Mathematical Software (ACM/TOMS)

The ACM/TOMS journal [3] is a forum for disseminating progress in the design and implementation of mathematical software. TOMS has a strong emphasis on making sure that the software discussed in a paper exists in a tangible way. Although no formal policy explicitly lists exactly what is acceptable, TOMS is a high-impact journal because many of the articles present software that eventually becomes heavily used by the research community.

Other journals regularly publish computational results, and some have a similar concrete relationship to released software, but in our experience many of computational journals do not have any concrete process to confirm that the software used to generate result is working properly, or available to others, much less that the results are accurate.

3.3 Executable Paper Grand Challenge

Elsevier recently announced its Executable Paper Grand Challenge [11]. This contest was created to make the publication of computational science results higher quality and easier. The specific questions the challenge asked were:

- How can we develop a model for executable files that is compatible with the user's operating system and architecture and adaptable to future systems?
- How do we manage very large file sizes?
- How do we validate data and code, and decrease the reviewer's workload?
- How to support registering and tracking of actions taken on the 'executable paper?'

The contest organizers recognize that the data, software and other elements that go into producing a computational science paper are often lost, since only the paper is typically archived.

4. CHALLENGES

Although any independent verification effort of scientific results is challenging, it seems to us that verifying computational results is particularly hard for several reasons, and will require years of effort, changes in publishing processes and deep cultural changes.

4.1 Complexity of Computing Environment

One of the most daunting challenges for reproducing results is the complexity of our computing environments. Although reproducing the numerical values generated by a code may be relatively straight-forward, assuming access to the original software environment, many CSE papers also include timing results. Timing can be impacted by any level of the computing platform: Processor choice, operating system, runtime libraries and compiler flags. Reproducing the exact configuration is very challenging, especially after the initial results are produced, even if done by the same user.

4.2 Cost

Independent verification is costly. It requires investment in infrastructure by the research team and the independent verifier. It can also put a lot of demand on the verifier, especially if the reference computing system environment is not easily available. Presently journal referees are not expected, nor have the facilities, to assess the quality of computational results beyond examining the given manuscript. External verification is possible, by looking for presence of the software on the Internet, but this is an ad hoc approach that, even if it does not find the software, does not typically disqualify the paper.

Another cost is the likely increase in time to produce a manuscript for submission to a journal, as well as the time between submission and publication. However, we expect that the quality of an article will improve if reproducibility and independent verification is required, and the results of published articles can be more highly trusted.

4.3 Cultural

Perhaps the most difficult challenge we face in making progress toward reproducible results and independent verification is cultural. Most computational scientists do not have the habit or desire to share their computing environment with verifiers. This reluctance can be overcome, probably most effectively by funding agencies including independent verification as a requirement for funding.

5. USEFUL ACTIVITIES

As mentioned in Section 3, there are existing efforts whose success move us in the direction of reproducible results and independent verification. However, there are more activities that can be emphasized. Some of them are emerging only now and provide us with new opportunities.

5.1 Standardized Platforms

A very helpful tool toward our goals would be a standard, widely available computing platform. Since many computer systems are based on commodity parts, operating systems and compiling environments, it has been possible for some time to have independent verification platforms. However, there are so many possible combinations that practically speaking it is very hard to expect an independent verifier to be able to confirm results on a separate system.

Recently "cloud" computing has become a real commercial service. As one example, Amazon's Electronic Cloud Computing (EC2) service [1] provides a high-performance computing platform that is available to anyone on the Internet, even providing a base number of free computing hours. This service can provide a legitimate approach to independent verification, by having standard platforms available, even established by particular journals. Perhaps, just like each journal has its own LaTeX format templates, each computational journal can have its own cloud computing platform, and porting to that platform is one way to provide access to an independent verifier.

This approach will certainly not address all needs. For example, some computational results are of interest because they are generated on the latest and best computing platforms. In those situations, it is perhaps possible to allow the reviewer to obtain access to the same system and to directly access the research team's software environment.

5.2 Use Standard Libraries

In addition to a common platform, use of standard libraries, perhaps also part of the journal cloud platform, could reduce variability in results. Specifically, the platform could include a suite of common libraries such as BLAS [15, 9, 10], LAPACK [2], FFTW [12], PETSc [6, 5, 4] and Trilinos. This approach would also encourage use of standard libraries in general, which is also a good thing.

5.3 Journal Policies

A key to making progress on reproducible results and independent verification has to be journal policies, if only to have the computational science journals include some form of interrogation about the availability and usability of the software used by a paper. Ideally, we would like journals to require independent verification of results prior to publication, as is done in other areas of science. Even though such rigor is not possible today, we can at least modify journal policies to make progress in this direction.

5.4 Funding Agencies

The final key to progress is to engage funding agencies and encourage them to expect funded research teams to produce results that are reproducible and independently verifiable. If CSE research teams get funding only by agreeing to reproducibility requirements, in a similar way to the ASCI program mentioned in Section 3, we believe that the cultural and infrastructure requirements will be naturally addressed by teams without imposing any specific formal SQA/SQE requirements.

6. CONCLUSIONS

CSE has often been described as the third fundamental paradigm for scientific advance, and it is truly recognized as a valuable pursuit, even being mentioned in the past two State of the Union addresses by US president Barack Obama. However, as a scientific process, the lack of reproducibility and independent verification is noticeable, as is the lack of software engineering rigor. We believe that these two weaknesses are linked, and that demanding reproducibility of computational results will positively impact adoption and innovative adaptation of software engineering practices and processes in CSE. We have already seen this impact in some projects, such as the DOE ASCI program. The availability of cloud computing and a broad collection of standard libraries, and the awareness of journal editors and funding agencies make it a good time to raise awareness of reproducibility as a positive mechanism for improving the quality and impact of computational science.

7. REFERENCES

[1] Amazon.com. Amazon elastic compute cloud (amazon ec2), 2011. `http://aws.amazon.com/ec2`.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM Pub., Philadelphia, PA, second edition, 1995.

[3] Association for Computing Machinery. Transactions on mathematical software, 2011. `http://toms.acm.org`.

[4] S. Balay, W. Gropp, L. McInnes, and B. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhauser Press, 1997.

[5] S. Balay, W. Gropp, L. McInnes, and B. Smith. PETSc 2.0 users manual. Technical Report ANL-95/11 - Revision 2.0.22, Argonne National Laboratory, 1998.

[6] S. Balay, W. Gropp, L. McInnes, and B. Smith. PETSc home page. `http://www.mcs.anl.gov/petsc`, 1998.

[7] J. Carver. Fourth international workshop on software engineering for computational science and engineering., 2011. `http://www.cs.ua.edu/%7ESECSE11/`.

[8] Department of Energy. Advanced simulation and computing, 2011. `http://www.sandia.gov/nnsa/asc`.

[9] J. Dongarra, J. DuCroz, S. Hammarling, and R. Hanson. An extended set of Fortran basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 14, 1988.

[10] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, March 1990.

[11] Elsevier. Executable paper grand challenge., 2011. `http://www.executablepapers.com/`.

[12] M. Frigo and S. G. Johnson. Fftw 3.2.2., 2011. `http://www.fftw.org/fftw3_doc`.

[13] M. Heroux, R. Bartlett, V. H. R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, and A. Williams. An Overview of Trilinos. Technical Report SAND2003-2927, Sandia National Laboratories, 2003.

[14] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.

[15] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software*, 5, 1979.

[16] National Science Foundation. Software infrastructure for sustained innovation, 2011. `http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503489&org=NSF&sel_org=XCUT&from=fund`.