DOE Software Engineering Challenges in Reliable Ecosystems, Portability, and Reproducibility

Authors: Miranda Mundt, Reed Milewicz, Evan Harvey, Wade Burgess, Andrew Younge, Jay Lofstead

Sandia National Laboratories

Corresponding Author: Miranda Mundt (mmundt@sandia.gov)

Introduction

The lead authors of this response are members of the Department of Software Engineering and Research at Sandia National Laboratories, with supporting contributions from the Department of Scalable System Software. Our diverse team focuses on advancing the study and practice of software engineering to support scientific software. Within the context of the Exascale Computing Project (ECP), we have provided in-depth support to ECP-funded software projects at Sandia and elsewhere; this has included (1) performing embedded development work, (2) building and maintaining cyberinfrastructures, (3) conducting fundamental and applied research in software engineering, and (4) offering consultation and training on development best practices.

The race to exascale has proven influential on the practice of scientific software development within the DOE. The shift towards increasing heterogeneity in computing architectures, a more interdependent software ecosystem, and multi-disciplinary and multi-institutional collaboration have all necessitated changes in how scientific software is created and maintained. This evolution in ways of working with scientific software is still ongoing, however; as we look to the future, we anticipate many challenges on the path to a more robust and mature HPC ecosystem. For our part, our team recognizes that the quality of software depends upon the quality of its development. In other words, by improving the means of production -- teams and their technologies and practices -- we can improve the resulting software. Achieving a culture of effective stewardship requires investing in the tools, infrastructures, and practices that will enable that culture.

To that end, we have identified three key challenge areas where progress could enable more effective stewardship: (1) ecosystem sustainability, (2) performance portability, and (3) reproducibility. Each poses risks that, left addressed, could endanger the DOE's investments in scientific computing; by the same token, each presents opportunities for innovation to fundamentally improve the development and use of HPC scientific software.

Sustainable Software Ecosystem

In their 2009 article, Bosch discusses the trend in software engineering from a simple product line to an increasingly complex software ecosystem (Bosch 2009). They point out two reasons why a company might adopt a software ecosystem: limited internal capability (e.g., an external solution exists already, or it would be economically infeasible to create an internal solution) and user customization requirements (e.g., the ability to create a unique configuration per each user's needs).

These two use cases are not absent from DOE national laboratories, and labs already engage to some extent in the larger software ecosystem of scientific programs across the world. Lab developers use open-source and commercial compilers (e.g., GCC, Intel) rather than creating their own; they use popular third-party libraries (e.g., NetCDF, Boost) for their well-known and accepted solutions to scientific programming needs.

This participation in the larger scientific programming software ecosystem has up to this point been a passive effort. Developers identify and adopt external packages as they see fit for their project's needs, many times downloading and installing whatever version seems appropriate at their current stage of development. While this method can be acceptable in the short-term, the long-term implications must be considered.

Let us imagine a project named ALPHA. At the beginning of the ALPHA project, there are two developers working closely together in their programming efforts. They share knowledge directly, make decisions together, and generally stay in sync throughout the early stages of ALPHA's development lifecycle. Several years later, the ALPHA software becomes desirable and gains stakeholders and external users. As a result, more developers join the ALPHA team. The two original developers, not anticipating the growth and popularity of ALPHA at its onset, do not have in-depth developer documentation regarding environment and version requirements for compilers, MPIs, and third-party libraries. The new developers must spend a non-negligible amount of time to reproduce a development environment that matches that of the original team members.

How much time is lost in this scenario by a lack of a sustainable, quickly reproducible software ecosystem? The answer to this question varies widely. As Howison et al. found, scientific end-users (that is, scientists engaged in domain science who either use or develop code as a means to pursue scientific innovation) have a vested interest in what components were used to work with a code throughout its history as well as software that is well-understood and recognized by other domain scientists (Howison et al. 2015).

In a survey aimed at specifically geospatial software users, Vandewalle et al. discovered that the existing software ecosystem has multiple barriers to entry in the form of ease of use, availability and access to tools and data, and instability of existing tools (Vandewalle et al. 2021). The introduction of a sustainable, well-managed and maintained software ecosystem which can provide common toolchains and preserve historical components is quintessential to the reliability and longevity of our scientific models and algorithms. Steps towards this goal

have been taken through package managers such as Spack, which aims to provide a single entry point for a variety of operating systems and microarchitectures (Gamblin et al. 2015). Spack has been explored by multiple national labs, but generally in a silo rather than as an overarching effort. Even within a single community like Sandia National Laboratories, four distinct teams, the Department of Software Engineering and Research included, are attempting to use Spack individually to provide software toolchains for scientific software developers, often having to develop their own wrappers and extensions, which means there is four times as much spending on incorporating the tool into projects than is truly necessary. These efforts have remained separate partially due to general lack of overarching goals and directives from software ecosystem architects and partially due to differing opinions on the best delivery system for toolchains, as well as the necessary toolchains for development projects.

In the realm of sustainable software ecosystems, there is plenty of work to be done. Through common documentation standards and supporting tools, each lab can develop locally useful toolchains that could more easily be incorporated with those from other institutions or teams. To achieve this, a standard implementation or interface could be adopted or developed. This could be an agreement to rely on Spack as the toolchain deployment tool, cloud instances of virtual machines and containers, etc. Alongside this standardized release process, careful, required documentation would enable better understanding at a project-level of requirements, inputs, and outputs of scientific software. Ultimately, the standardized approach to supplying toolchains manages the interoperability complexities between each lab as well as the software ecosystem misalignments.

The introduction of a managed cross-lab software ecosystem, however, will then create the necessity for a larger software ecosystem governance effort. Respondents to the survey by Serebrenik and Mens relating to challenges in software ecosystem research stated that the future evolution of an established ecosystem is particularly of concern since this aligns business goals with technical concerns (2015). A concentrated governance effort will help ensure that the individual ecosystems align harmoniously and continue to enable more seamless collaboration between labs.

We can also consider how to build sustainability into this ecosystem from the ground up. Each individual project must be accountable for developing software of sufficient quality for reuse. Authors of this response have contributed to the creation of the Productivity and Sustainability Improvement Planning (PSIP) process, which is a lightweight toolkit aimed at helping projects identify areas of improvement and developing incremental plans to address those issues (Heroux et al. 2019). A useful implementation of PSIP would be to improve software ecosystem documentation at a team level. That is, special focus could be given to the documentation of necessary packages and environment variables for a particular software's ability to run. More development on PSIP specifically geared at incorporating software ecosystem concerns would be a step in the right direction at empowering project leads in this area.

Portability

Portability is a risk area for the DOE software ecosystem. Whereas the need for portability solutions within HPC has continued to grow in recent years, it has steadily faded into the background of mainstream software development. End-user computing environments are much more standardized and developer-friendly than in decades past, and the shift towards cloud computing and containerization have dramatically reduced the costs of providing customized and networked application environments. This is directly visible in the research literature on portability, as publications by non-HPC researchers make up a dwindling of the overall literature on portability. By proxy, this means that less investment from mainstream industry and attention from academia is going into tools, techniques, and infrastructures for application portability, leaving the scientific software community to fend for itself. This lack of focus on repeatable methods to ensure portability puts current high-performance code at risk. Especially in the case of next-generation supercomputers, existing code built on existing architectures is not guaranteed to function within a new, state of the art architecture and environment.

To the extent that the HPC community can align itself with development in industry, we can do so to make gains on portability. Cloud-computing-style server setups, for instance, allow access and rapid deployment of identical machines, particularly when paired with deployment automation tools like Ansible (Hochstein et al 2017). Meanwhile, containers can provide the complete environment necessary in order to run an application (Anderson 2015). Case in point, we are working to roll out support for ready-made containers for application developers and users within our research center. Containers have a rich variety of uses, from facilitating continuous integration and deployment, providing portable environments for developers and users across different institutions (Kurtzer et al. 2017, Bam 2018), enabling seamless deployment to HPC clusters (Abraham et al. 2020), and enhancing the portability of complementary solutions like scientific workflows (da Silva Alves and Charão 2020). Adoption of containers-as-a-service implies maintenance costs and ongoing stakeholder engagement to ensure that that service meets user needs. There also needs to be sufficient interoperability of container portability solutions between DOE institutions, which entails some degree of software ecosystem governance. Containers are an example where a DOE stewardship effort could show leadership by helping to build consensus and coordinate solution development.

Solutions like containers, however, can only take us so far. On the tools and libraries front, our team actively contributes to efforts supporting application portability, including Kokkos, KokkosKernels, Spack, and xSDK. Labor-saving innovations like performance portability abstractions and cross-platform package distribution tools ease the burden of moving software stacks across hardware environments, thus opening the door for more achievable sustainability in software ecosystems. A significant amount of work, however, is needed to ensure those solutions are both used and useful. In the case of Kokkos and KokkosKernels, this has included not just algorithm development but also project management more broadly, such as by standing up continuous integration testing, code style checks, and automating the generation of documentation. Developing portability solutions for use by the wider community requires a holistic approach to all aspects of the software development to meet demand at scale.

Reproducibility

Reproducibility in science is the principle that different researchers should be able to arrive at the same results from an experiment or observational study using the same methodology. This is the cornerstone of the scientific method: results need to be reproducible to be considered scientific knowledge. Moreover, the need for reproducibility pushes researchers to approach their experiments with sufficient care and rigor, leading to higher quality work. In the context of computational science, reproducibility means the ability to "to carry out tasks that are equivalent in substance to the original but may differ in ways that are not expected to be significant to the final result," that is, robust enough that different teams potentially working with different initial artifacts can arrive at equivalent results (Ivie et al. 2018).

In scientific computing, the statistics regarding reproducibility leave much to be desired. Ivie and Thain report results on several studies that explored reproducibility in scientific software: a 10% success rate from 53 articles in biotech and a 21% success rate in pharmaceuticals. It is no wonder that the situation appears so dire. An average scientist in the process of developing scientific software have to record or remember hardware and software setups such as: which version of a dependency was installed, how the software was configured, which computing node was used, if any changes have been applied to that node since its last use, etc. With so many potential points of failure, who is to say the results can be trusted?

Trust in our systems is essential. At Sandia National Laboratories, we rely on simulation to understand key mission areas such as energy, climate change, national security applications, and more. When simulation results differ from one run to the next, we need solid protocols and processes in place to determine which run was incorrect and how to fix or avoid the root cause in the future. Only with strong reproducibility standards in place to enable root cause analyses can we have trust in our software systems and our results.

Software defects are a leading cause to retractions of scientific papers, which has a downstream effect on any future work which was built upon those original results (Miller 2006). This is especially important for DOE national labs, as our peace, prosperity, and security increasingly depend upon the quality and trustworthiness of computational results. Better reproducibility will enable computational scientists to iterate upon their work more rapidly and efficiently and that of others, accelerating the pace of innovation. That is, the more we can do to improve reproducibility, the more useful and impactful computational science can be.

This is in part a training challenge. The average researcher has little or no formal training in software development best practices that could support reproducibility. Case in point, Sufi et al. have identified opportunities where adoption of tools could help researchers meet reproducibility objectives, including modern version control, data sharing platforms, containers, and notebook and workflow systems (Sufi et al. 2014). At Sandia National Laboratories, a prime example would be in automation for continuous integration (CI). While CI testing can help ensure reproducibility as software evolves, the adoption and effective use of these technologies is held back by incompatible syntaxes and standards among tools and mismatches in tool

capabilities and use cases (e.g., CI testing across multiple repositories, in different security contexts, etc.). Currently, no single tool meets the use-case of all scientific software in development or production, and there is a lack of good guidance for researchers on which tools to use, when to use them, and how to use them well. This is an area where DOE stewardship could support the development of training materials. For our part, as standards and expectations for reproducibility continue to mature, we anticipate assisting scientific software teams with ensuring the reproducibility of their software by providing such resources. Related to this, various authors have called attention to how reproducibility-enhancing practices could be incorporated throughout the software lifecycle, from requirements engineering (Smith et al. 2019) to testing (Stodden and Krafczyk 2018) to maintenance (Leman et al. 2021). Again, these are areas where training could help to promote reproducibility-enhancing practices.

Meanwhile, the growing trend of integrating Research Software Engineers (RSEs) into scientific software projects presents novel avenues for improving the state of practice around reproducibility. RSEs are helping to bridge the gap between software engineering and computational science as professionals who are capable of understanding the terminology, needs, and best practices in both realms and the way in which to marry the goals of both domains (Storer 2017). Providing direct software engineering support for researchers and their software has been a explicit goal of our department since its formation in 2019 (Milewicz et al. 2020). However, a capability gap is a lack of evidence-based SE guidance on reproducibility as a software quality, such as data on trade-offs between reproducibility other desirable qualities or formal and empirical models linking software design and implementation decisions to reproducibility outcomes. Closing that gap by deepening our understanding of engineering software for reproducibility will be a high priority moving forward. On our own team, we have made some efforts in this direction, such as by conducting replication studies to better understand reproducibility needs (e.g., Willenbring 2015). We hope to do future work in a similar vein, as well as experience reports and case studies on achieving reproducibility in the wild. This is an area where researchers could offload work to RSEs, allowing them to focus on the core physics and engineering while RSEs investigate strategies for improving system reproducibility.

A dedicated RSE alone may not be enough, though. The instruction and adoption of development methodologies - particularly Agile and the application of its principles - may also be key to resolving reproducibility concerns (Beck et al. 2001). Particularly, we will speak to four of the principles:

- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- The best architectures, requirements, and designs emerge from self-organizing teams.

Agile processes such as Scrum or Kanban are categorized by their formality and levels of rigor, which are requisite to the promotion of sustainable development. Sustainable development is not possible, however, unless members are motivated to work on the project. The same truth directly relates to the remaining two Agile principles. Motivated individuals are more likely to pay continuous attention to technical excellence and good design, which in turn leads to sustainable reproducibility. Furthermore, motivated individuals that comprise a team will be more effective at self-organization, resulting in the emergence of the best architectures, requirements, and designs, upon which sustainable reproducibility relies. While an integrated RSE brings the expertise, the cooperation and motivation of all participants in the reproducibility process will lead to its ultimate success.

Conclusion

We identified three key challenges areas where progress could enable more effective stewardship. First, we discussed the enablement of a sustainable software ecosystem. We argued that a lab-wide standardized approach would enable more seamless collaboration, which is achievable through better tooling and documentation as well as a guiding governance effort. Existing tools like Spack the package manager and PSIP for team process improvement can be utilized to achieve better ecosystem organization and governance. Second, we turned our attention to portability and its challenges within the world of HPCs. We spoke to containers and tools and libraries as possible solutions to portability challenges with more than just an emphasis on the tools themselves. Efforts must also include project management, software quality efforts, and documentation. Last, we focused on reproducibility. In this challenge area, we argued that trust in our systems and our results is essential for innovation. To this end, better training must be made available to research scientists on available resources and tools to enhance reproducibility. Additionally, dedicated research software engineers should be integrated into software teams to bring software engineering expertise. All team members, however, must have the proper motivation to contribute to the reproducibility process.

Please direct any questions or comments about this RFI response to Miranda Mundt (email: mmundt@sandia.gov). Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DENA0003525. SAND-NUMBER-HERE.

References

- Hochstein, L., & Moser, R. (2017). Ansible: Up and Running: Automating configuration management and deployment the easy way. " O'Reilly Media, Inc.".
- Anderson, C. (2015). Docker [software engineering]. Ieee Software, 32(3), 102-c3.
- Kurtzer, G. M., Sochat, V., & Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. PloS one, 12(5), e0177459.
- Bam, Bikram. Research as code: Instrumenting scientific computing as executable containers. MS thesis. 2018.

- Abraham, S., Paul, A. K., Khan, R. I. S., & Butt, A. R. (2020, October). On the use of containers in high performance computing environments. In 2020 IEEE 13th International Conference on Cloud Computing (CLOUD) (pp. 284-293). IEEE.
- da Silva Alves, Bruno, and Andrea Schwertner Charão. "Towards Integration of Containers into Scientific Workflows: A Preliminary Experience with Cached Dependencies." International Conference on Computational Science and Its Applications. Springer, Cham, 2020.
- Ivie, P., & Thain, D. (2018). Reproducibility in scientific computing. ACM Computing Surveys (CSUR), 51(3), 1-36.
- Baxter, R., Hong, N. C., Gorissen, D., Hetherington, J., & Todorov, I. (2012, September). The research software engineer. In Digital Research Conference, Oxford (pp. 1-3).
- Storer, T. (2017). Bridging the chasm: A survey of software engineering practice in scientific programming. ACM Computing Surveys (CSUR), 50(4), 1-32.
- Miller, G. (2006). Scientific publishing: A scientist's nightmare: Software problem leads to five retractions. Science 314, 5807 (2006), 1856–1857.
- Milewicz, R., Willenbring, J., & Vigil, D. (2020). Research, Develop, Deploy: Building a Full Spectrum Software Engineering and Research Department. arXiv preprint arXiv:2010.04660.
- Sufi, S., Hong, N. C., Hettrick, S., Antonioletti, M., Crouch, S., Hay, A., ... & Parsons, M. (2014, June). Software in reproducible research: advice and best practice collected from experiences at the collaborations workshop. In Proceedings of the 1st ACM sigplan workshop on reproducible research methodologies and new publication models in computer engineering (pp. 1-4).
- Stodden, Victoria, and Matthew S. Krafczyk. "Assessing reproducibility: An astrophysical example of computational uncertainty in the HPC context." The 1st Workshop on Reproducible, Customizable and Portable Workflows for HPC, HPC18. 2018.
- Smith, Spencer, Malavika Srinivasan, and Sumanth Shankar. "Debunking the myth that upfront requirements are infeasible for scientific computing software." 2019 IEEE/ACM 14th International Workshop on Software Engineering for Science (SE4Science). IEEE, 2019.
- Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... & Thomas, D. (2001). Manifesto for agile software development.
- Gamblin, T., LeGendre, M., Collette, M. R., Lee, G. L., Moody, A., De Supinski, B. R., & Futral, S. (2015, November). The Spack package manager: bringing order to HPC software chaos. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1-12).
- Howison, J., Deelman, E., McLennan, M. J., Ferreira da Silva, R., & Herbsleb, J. D. (2015). Understanding the scientific software ecosystem and its impact: Current and future measures. Research Evaluation, 24(4), 454-470.
- Vandewalle, R. C., Barley, W. C., Padmanabhan, A., Katz, D. S., & Wang, S. (2021). Understanding the multifaceted geospatial software ecosystem: a survey approach. International Journal of Geographical Information Science, 35(11), 2168-2186.

- Bosch, J. (2009, August). From software product lines to software ecosystems. In SPLC (Vol. 9, pp. 111-119).
- J. M. Willenbring, “Replicated computational results (RCR) report for BLIS: A framework for rapidly instantiating BLASx functionality,” ACM Trans. Math. Softw., vol. 41, no. 3, Sep. 2015.
- Heroux, M. A., Gonsiorowski, E., Gupta, R., Milewicz, R., Moulton, J. D., Watson, G. R., ... & Raybourn, E. M. (2019). Lightweight Software Process Improvement Using Productivity and Sustainability Improvement Planning (PSIP). In Tools and Techniques for High Performance Computing (pp. 98-110). Springer, Cham.