SAND2019-1048C

# Position Paper: Towards Usability as a First-Class Quality of Scientific Software

Reed Milewicz
Sandia National Laboratories
Email: rmilewi@sandia.gov

Paige Rodeghero
Clemson University
Email: prodegh@clemson.edu

***Abstract*—The modern scientific software ecosystem is instrumental to the practice of science. However, software can only fulfill that role if it is readily usable. In this position paper, we discuss usability in the context of scientific software development, how usability engineering can be incorporated into current practice, and how software engineering research can help satisfy that objective.**

## I. Introduction

The future of the scientific enterprise requires a sustained, robust, and reliable ecosystem of scientific software. This is necessary to meet the ever-growing demands for scalable simulation and data analysis, with some authors suggesting that we are moving towards a paradigm of science that is equal parts computational, empirical, and theoretical [1]. While progress has been made, software as an instrument has not yet reached a level of maturity comparable with the more conventional tools of empirical and theoretical science. A 2016 report by the National Strategic Computing Initiative (NSCI) argued the current "ecosystem of software, hardware, networks, and workforce is neither widely available nor sufficiently flexible to support emerging opportunities" [2]; the strategic plan highlighted the need for "a portfolio of new approaches to dramatically increase productivity in the development and use of parallel HPC applications" as a focus for future research.

It is clear that the demand for scientific software can no longer be met by individuals working in isolation, and fostering more effective collaboration is a necessity [3]. Modern high-end scientific computing applications rely on complex software stacks assembled from an ecosystem of software packages developed by many teams across different disciplines. For example, even a barebones PDE code may draw upon distinct libraries for meshing, discretization, linear solves, post-processing, and visualization. Researchers would not be able to affordably develop their application codes without the support of community software, but using other's code is an exercise in trust, trust that the code can perform its intended function both now and in the future. Unfortunately, scientists frequently use (and misuse) software without understanding how that software actually works [4], and acquiring sufficient understanding is time-consuming [5]. In other words, becoming a literate user of or contributor to a scientific software package carries opportunity costs, yet defering those costs poses a risk to informed trust and, consequently, the value that the software provides.

As we approach the exascale era and beyond, the scientific software community faces a crisis created by the confluence of disruptive changes in computing architectures and demands for greatly improved simulation capabilities. This crisis brings with it a unique opportunity to fundamentally change how scientific software is designed, developed, and supported. This is especially true for software developed at US national laboratories. Keyes et al. 2013 observes that present and future objectives of the DOE require multiphysics solutions that bring together many different codes, disciplines, and institutions [6]; accordingly, the need for performance and correctness must be balanced against the need for ease-of-use across increasingly complex software stacks. For this reason, we argue that **usability** must become a first-class software software quality moving forward. By usability, we mean the capability of the software product to be understood, learned, used and attractive to the use, when used under specified conditions [7]. Easing the pathway to proficiency by making the software more accessible and understandable enables users to make better use of that code and to be more confident in the results.

## II. Background

We would like to have scientific software that is correct, performant, and usable, yet none of these qualities happen by accident: they require intentional planning and action. Moreover, usability, like any non-functional software quality, is not an absolute good. Scientific software is necessarily complicated because the problems they solve are also complicated; extensive expertise is needed both to write the code and to use the code. A more usable code may be less capable, less performant, or less portable. Moreover, deciding to improve a non-functional quality requires an investment of time and money, and those resources may be spent more effectively elsewhere. Indeed, qualities such as correctness must come first, but we argue that better trade-offs could be made and that usability must be included in that conversation.

Usability engineering is well-practiced in the conventional software industry, but it currently represents "the most ig-

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 DE-NA0003525. SAND-XXXX.

nored and unattended phase of scientific software solution development" [8]; techniques for studying usability are not as prevalent or mature among scientific software developers as those that measure correctness or performance. Likewise, research into usability engineering for scientific software is still nascent (see [9],[10]). This is where the software engineering research community can be of great help, by identifying and demonstrating the effectiveness of tailored strategies for creating usable scientific software.

While usability is a topic that concerns all stages of the software lifecycle, the long lifespan of HPC research codes means that our projects of interest are usually "in the middle". For example, within the US Exascale Computing Project, the median age of an application code is 7 years. An emphasis must be placed on tools and techniques that are (1) well-defined, (2) incremental, and (3) accommodating of the reality that projects have already made significant commitments in their design and implementation. This is a necessary step towards improving the current state of practice as well as making the case for funding models that support such activities.

## III. Roadmap

We now present a roadmap for usability research that satisfies those objectives in the short-to-medium term. Usability engineering is a sufficiently broad and deep topic that ours is not meant to be exhaustive. For example, given the diversity of scientific software, there is a need for more exploratory studies using surveys and ethnographic techniques ala [9]. Likewise, the community would also benefit from research that covers usability across the full software lifecycle in the vein of [10].

Our focus, however, is on concrete tools and strategies that could be immediately applied in the form of team policies and project deliverables, in particular usability evaluation methods (UEMs). Borrowing from Fernandez et al., we define UEMs as "procedures composed by a series of well-defined activities to collect data related to the interaction between the end user and a software product, in order to determine how the specific properties of a particular software contribute to achieving specific goals" [11]. We have identified two challenge areas where such UEMs could be applied:

**Source Code and APIs**: Scientific software users are often end-user developers, drawing upon libraries written by others to create their own codes. Much of their time is spent reading and navigating source codes and APIs to construct a mental model of the software that they use. The question then is whether the mental model which the code encourages is one that is effective for the user and satisfies their needs. This is a two-way process: a code may be ostensibly well-written (by some measure) yet fail to provide the information that user requires, or may put unstated cognitive demands on the user. Potential solutions include:

- Inspection-based guideline reviews and heuristic evaluation methods for assessing the readability of scientific source code.
- (Semi-)automated software metrics for API usage pattern complexity that are applicable to scientific software.
- Task-based interview strategies for user testing.

**Learning Resources**: Studies have shown that users benefit from having access to a variety of different learning materials, and that programmers tend to be highly opportunistic in their work, engaging in just-in-time learning and adapting existing resources to solve new problems. Modern projects have a wealth of options at their disposal including tutorials, example codes, Q&A repositories, documentation, and training. But what is the most cost-effective way of creating and maintaining these resources to satisfy that need? Potential solutions include:

- Needs-based assessments with participation of stakeholders.
- Cognitive walkthroughs, in particular guidelines for creating personas to model resource usage.
- Developer-driven protocols for reviewing resource and quality coverage.

## IV. Conclusion

The value of scientific software is intimately tied to its usability, the ability to pick it up and put it to work answering a scientific question. However, fitting usability engineering into the current state of practice remains an open challenge. For this reason, our position paper serves as a call to action and an invitation for dialogue among software engineering researchers.

## References


[1] G. Bell, T. Hey, and A. Szalay, "Beyond the data deluge," *Science*, vol. 323, no. 5919, pp. 1297–1298, 2009.

[2] J. P. Holdren and S. Donovan, "National strategic computing initiative strategic plan," National Strategic Computing Initiative Executive Council Washington United States, Tech. Rep., 2016.

[3] M. J. Turk, "Scaling a code in the human dimension," in *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*. ACM, 2013, p. 69.

[4] L. N. Joppa, G. McInerny, R. Harper, L. Salido, K. Takeda, K. O'hara, D. Gavaghan, and S. Emmott, "Troubling trends in scientific software use," *Science*, vol. 340, no. 6134, pp. 814–815, 2013.

[5] J. Y. Monteith, J. D. McGregor, and J. E. Ingram, "Scientific research software ecosystems," in *Proceedings of the 2014 European Conference on Software Architecture Workshops*. ACM, 2014, p. 9.

[6] D. E. Keyes, L. C. McInnes, C. Woodward, W. Gropp, E. Myra, M. Pernice, J. Bell, J. Brown, A. Clo, J. Connors *et al.*, "Multiphysics simulations: Challenges and opportunities," *The International Journal of High Performance Computing Applications*, vol. 27, no. 1, pp. 4–83, 2013.

[7] "International standard iso/iec 9126-1. software engineering – product quality – part 1: Quality model," International Organization for Standardization / International Electrotechnical Commission, Geneva, CH, Standard, 2001.

[8] Z. Ahmed, S. Zeeshan, and T. Dandekar, "Developing sustainable software solutions for bioinformatics by the butterfly paradigm," *F1000Research*, vol. 3, 2014.

[9] D. Sloan, C. Macaulay, P. Forbes, and S. Loynton, "User research in a scientific software development project," in *Proceedings of the 23rd British HCI Group Annual Conference on People and Computers: Celebrating People and Technology*. British Computer Society, 2009, pp. 423–429.

[10] L. Ramakrishnan and D. Gunter, "Ten principles for creating usable software for science," in *e-Science (e-Science), 2017 IEEE 13th International Conference on*. IEEE, 2017, pp. 210–218.

[11] A. Fernandez, E. Insfran, and S. Abrahão, "Usability evaluation methods for the web: A systematic mapping study," *Information and software Technology*, vol. 53, no. 8, pp. 789–817, 2011.