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

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

Paige Rodeghero
Clemson University
Email: prodegh@clemson.edu

***Abstract*—The modern HPC 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. 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.

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-2019-1048C.

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 US Department of Energy 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 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.

Usability engineering is well-practiced in the conventional software industry, but it currently represents "the most ignored 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], and [11]). 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.

II. Open Questions; Promising Directions

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. Our focus therefore 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 a UEM as "a procedure which is composed of a set of well-defined activities for collecting usage data related to end-user interaction with a software product and/or how the specific properties of this software product contribute to achieving a certain degree of usability." [12]. Topics of interest include:

- The mainstay of UEMs is **user testing**, where evaluators observing participants interacting with software in order to identify usability issues. These include thinking-aloud protocols (where the user describes their actions) and co-discovery learning (where pairs of users must work together to complete a task) [13]. While user testing is a tried-and-true method, it is an expensive and time-consuming process, and it isn't yet clear about how user testing fits into the scientific software workflow. How and when should user testing be employed, and under what conditions will user testing be most effective?
- Based on techniques in cognitive psychology, a **cognitive walkthrough** (CW) method consists of evaluators taking on the role of a user and stepping through interactions with the product, and enabling evaluators to critique learnability and memorability of a software system [14]. This is most often accomplished through the use of personas, which are detailed models of users that factor in their background knowledge, motivations, and attitudes. CWs are advantageous because they can be applied at any stage of the life cycle, and do not require user participation, but they do require accurate and informative user models. How do we take what we know about scientific software developers and turn it into an actionable model? Are these models generalizable across disciplines, or would they need to be tailored to a particular application domain?
- Checklists for **heuristic evaluation** consist of holistic statements about the desirable properties of a system and how they pertain to usability; as an example, Rusu et al. 2011 provides usability heuristics for grid computing frontends, such as the need to provide "shortcuts, abbreviations, accessibility keys or command lines for expert users," and showcases their application to real-world systems [15]. Checklists are low-hanging fruit in that they are easy to apply, but the contents have yet to be determined.
- Analytical modeling methods provide an engineering-based approach to predicting usability; this includes **task environment analysis**, creating a model that maps users' goals to interactions with the software system [16]. We argue that modeling use cases in a formal way gives developers more control over the usability of a scientific software system under evolution. For example, Dubey et al. describes the growth of the FLASH library, and the snowball effect of acquiring users and use cases across many different domains[17]. Each use case may involve different features, and there needs well-maintained, easy-to-follow pathway through the library for that use case. There is a need to adapt and prove such techniques in the context of scientific software development.

III. 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] M. List, P. Ebert, and F. Albrecht, "Ten simple rules for developing usable software in computational biology," *PLoS computational biology*, vol. 13, no. 1, p. e1005265, 2017.

[12] 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.

[13] J. Nielsen, *Usability engineering*. Elsevier, 1993.

[14] C. Wharton, "The cognitive walkthrough method: A practitioner's guide," *Usability inspection methods*, 1994.

[15] C. Rusu, S. Roncagliolo, G. Tapia, D. Hayvar, V. Rusu, and D. Gorgan, "Usability heuristics for grid computing applications," *Proceedings ACHI*, pp. 53–58, 2011.

[16] M. Y. Ivory and M. A. Hearst, "The state of the art in automating usability evaluation of user interfaces," *ACM Computing Surveys (CSUR)*, vol. 33, no. 4, pp. 470–516, 2001.

[17] A. Dubey, K. Antypas, A. C. Calder, C. Daley, B. Fryxell, J. B. Gallagher, D. Q. Lamb, D. Lee, K. Olson, L. B. Reid *et al.*, "Evolution of flash, a multi-physics scientific simulation code for high-performance computing," *The International Journal of High Performance Computing Applications*, vol. 28, no. 2, pp. 225–237, 2014.