Wickedness, Trust, and Sustainability

Reed Milewicz, Sandia National Laboratories
rmilewi@sandia.gov, @rmmilewi

The scientific community is experiencing two major trends with respect to software: (1) software is increasingly pervasive in all disciplines of science, and (2) the scale of that software – in virtually every sense of the word – is growing as well. This was echoed in a 2018 survey of scientist-developers by Pinto et al., who found that 82% of respondents felt that they were spending spending "more time" or "much more time" developing software than they did 10 years ago[1]. It has been argued that the future of science is equal parts computational, empirical, and theoretical[2], and that the demand for software can no longer be met by individuals working in isolation[3]. That naturally leads to the question of sustainability: can something so important and so central to scientific practice be made to endure?

It's worth noting that the problem of software sustainability is not unique to the scientific software community. Software has "eaten the world": modern life has come to depend on software in much the same way that it depends upon electricity, running water, or roads and bridges. However, our ability to engineer enduring software infrastructure has not yet reached a comparable level of maturity, with technology visionaries like Vint Cerf warning of a "a brittle and fragile future"[4]. These concerns have spurred a steadily growing movement within the software engineering community around sustainability, with the 2014 *Karlskrona Manifesto for Sustainability Design* being emblematic of the trend[5]. To be brief, the emerging consensus on software sustainability is that it is a ***wicked problem***.

The concept of a wicked problem is taken from the field of policy studies, formally introduced by researchers Rittel and Webber in 1973 in their foundational work titled *Dilemmas in a General Theory of Planning*[6]. Wicked problems are a class of problems for which there can be no single definition, no isolatable cause, no single solution, no stopping rule (that is, no "done" state), no ultimate test of the quality of a solution, and for which every instance is essentially unique and novel.

In this view, rather than looking at sustainability as a singular problem that can be "solved", sustainability is best understood as an ongoing state of affairs. It's a reflection of our priorities, practices, and policies, and it's a challenge that demands a full spectrum response to both the software and the culture that creates it. For this reason, the Karlskrona Manifesto divides sustainability into five dimensions: individual, social, environmental, economic, and technical. Moving the needle on scientific software sustainability means that we have to identify the dimensions where we can apply effort and get results. Here I will focus on sociotechnical axis; that is, the scientific software community (the social), the ecosystem of their software (the technical), and the interactions between the two.

At the start of this paper, I mentioned the trend towards increasing "scale" in scientific software, but I left the term open to interpretation. Scale in software operates on many

SAND2019-6926 C

different levels: the number of lines of code, the number of hardware and software elements involved, or the number of people developing it – to name a few[7]. But what does it mean for scientific software to be "large-scale"? Of all these interpretations, I believe the most salient measure is a sociotechnical one. To borrow from Moe et al., the hallmark of large-scale software is that *no one person can know everything*[8]. This is a threshold is reached much faster for scientific software than for software in other fields: extensive expertise is needed both to create and use it, and the scope of expertise involved increases dramatically in complex, collaborative projects.

Given that no one can know everything, using someone else's code is an exercise in ***trust***, trust that the code can perform its intended function both now and in the future. Science cannot happen without informed trust – how can we trust our results if we cannot trust our instruments? – and not having it puts a tax on the whole community. When scientists misuse software without understanding how it actually works[9], when a team decides to roll their own implementation of an algorithm rather than rely on a hard-to-use or fragile third party library, or when a decision maker is unduly limited in her ability to trust the conclusions of a simulation[10], we are paying that tax.

Sustainability is a wicked problem, and wicked problems demand multilateral interventions. I've identified one angle of attack that connects the social and technical dimensions of the problem; a sustainable future for scientific software requires a mature, trustworthy ecosystem and a community of practice that supports effective, cross-cutting collaboration on software. Directions for future work include the following:

- **Strategies to build more cohesive scientific software teams and organizations.** Raybourn et al. have advanced arguments for a Team of Teams approach to support integrative scientific software collaborations[11]. Along similar lines, Milewicz and Raybourn have suggested empowering knowledge brokers, cultivating organizational awareness, and encouraging integrative work as methods to overcome sociotechnical barriers[12]. These are just a few examples – there are a wealth of good ideas out there – but what we need now is to put these ideas to the test to discover what works well.

- **Methods to improve scientific software as a material for communication.** Gewaltig and Cannon have highlighted the distinction between research-ready and user-ready scientific software[13]; software can be ostensibly trustworthy yet too unintuitive or complicated for others to make good use of it. The value of scientific software is tied to its usability, that is, our ability to pick it up and put it to work answering scientific questions; to this point, works by List et al. and Ramakrishnan and Gunter have stressed the need for user engagement in scientific software development[14] [15]. Similarly, recent work by Milewicz and Rodeghero argues that usability engineering techniques, such as cognitive walkthroughs and task analyses, can help bridge that divide[16]. Given that developers are unlikely to find funding for usability work until they have well-defined methods, clear goals, and measurable outcomes, there is a need for more foundational research in this area.

Acknowledgements: A special thanks to fellow Sandians Mary Alice Cusentino and Jim Willenbring for taking the time to give feedback on the first draft of this white paper.

Works Cited

[1] Pinto, G., Wiese, I., & Dias, L. F. (2018, March). How do scientists develop scientific software? an external replication. In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 582-591). IEEE.

[2] Bell, G., Hey, T., & Szalay, A. (2009). Beyond the data deluge. Science, 323(5919), 1297-1298.

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

[4] Cerf, V. G. (2017). A brittle and fragile future. *Communications of the ACM*, *60*(7), 7-7.

[5] Becker, C., Chitchyan, R., Duboc, L., Easterbrook, S., Penzenstadler, B., Seyff, N., & Venters, C. C. (2015, May). Sustainability design and software: The karlskrona manifesto. In Proceedings of the 37th International Conference on Software Engineering-Volume 2 (pp. 467-476). IEEE Press.

[6] Rittel, H. W., & Webber, M. M. (1973). Dilemmas in a general theory of planning. Policy sciences, 4(2), 155-169.

[7] Northrop, L. (2013, May). Does scale really matter? ultra-large-scale systems seven years after the study (keynote). In 2013 35th International Conference on Software Engineering (ICSE) (pp. 857-857). IEEE.

[8] Moe, N. B., Šmite, D., Šāblis, A., Börjesson, A. L., & Andréasson, P. (2014, September). Networking in a large-scale distributed agile project. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (p. 12). ACM.

[9] Joppa, L. N., McInerny, G., Harper, R., Salido, L., Takeda, K., O'hara, K., ... & Emmott, S. (2013). Troubling trends in scientific software use. Science, 340(6134), 814-815.

[10] Weirs, V. G., Fabian, N., Potter, K., McNamara, L., & Otahal, T. (2013). Uncertainty in the Development and Use of Equation of State Models. *International Journal for Uncertainty Quantification*, *3*(3).

[11] Challenges and opportunities. The International Journal of High Performance Computing Applications, 27(1), 4-83.
Moulton, D., Raybourn, E. M., McInnes, L., & Heroux, M. A. (2018). Enhancing Productivity and Innovation in ECP with a Team of Teams Approach (No. SAND2018-3987C). Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[12] Milewicz, R., & Raybourn, E. (2018, October). Talk to Me: A Case Study on Coordinating Expertise in Large-Scale Scientific Software Projects. In 2018 IEEE 14th International Conference on e-Science (e-Science) (pp. 9-18). IEEE.

[13] Gewaltig, M. O., & Cannon, R. (2012). Quality and sustainability of software tools in neuroscience. *Cornell University Library*, 1-20.

[14] List, M., Ebert, P., & Albrecht, F. (2017). Ten Simple Rules for Developing Usable Software in Computational Biology. PLoS Comput Biol, 13(1), e1005265.

[15] Ramakrishnan, L., & Gunter, D. (2017, October). Ten principles for creating usable software for science. In 2017 IEEE 13th International Conference on e-Science (e-Science) (pp. 210-218). IEEE.

[16] Milewicz, R., &Rodeghero, P.. (2019, May). Position Paper: Towards Usability as a First-Class Quality of HPC Scientific Software. *2019 International Workshop on Software Engineering for Science*.