Towards Evidence-Based Practice in Scientific Software Development

How software engineering research can empower the scientific software community.

Reed Milewicz; *Sandia National Laboratories*

A whitepaper for the 2020 Collegeville Workshop on Scientific Software

Introduction

Increasing productivity in scientific software development is necessary to meet the ever-growing demand for computing in science. This is well-understood within the computational science and engineering (CSE) community in the US, where a report by the National Strategic Computing Initiative (NSCI) has warned that the current "ecosystem of software, hardware, networks, and workforce is neither widely available nor sufficiently flexible to support emerging opportunities"[1]. For this reason, the NSCI has called for "a portfolio of new approaches to dramatically increase productivity in the development and use of parallel HPC applications". Enabling productivity means reducing time-to-science and empowering teams to tackle ambitious challenges. In short, there is an urgent need for advances in this area to enable scientist-developers to do their jobs more efficiently and effectively.

The question of productivity demands systematic investigation and evidence-based intervention in the same way that our community approaches domain science or mathematics. Without sufficient understanding, we run the risk of solving the wrong problems, having the right solutions go unused, or failing to seize emerging opportunities. Fortunately, productivity has long been a focal point for software engineering research (also known as software science, see [2]), a field responsible for over five decades of scholarship on this subject.

In this paper, we argue that our community should leverage this wealth of resources through **evidence-based practice**. The evidence-based practice movement in software engineering advocates integrating current best evidence from research with practical experience and human values to improve decision-making related to software development and maintenance[3]. In this white paper, we'll examine what that means from a software science perspective, reflecting on our experiences within the Software Engineering and Research Department at Sandia National Laboratories.

Background

Software science is the study of software systems and their development, operation, and maintenance. Whereas the computational scientist sees software as a tool for science, the software scientist makes software the subject of science, applying the scientific method to model, understand, and predict the factors that lead to high-quality software systems. As an engineering science, this research tends to be motivated by practical problems and favors questions that lead to practical solutions. In this effort, the discipline has enjoyed considerable success: many of the tools, practices, and processes that are employed by software engineering professionals today trace their origins to research and the dynamic relationship between academia and industry.

A perennial subject of interest in software science is productivity, which has a rich history of scholarship dating back to the "software crisis" of the late 1960s and early 70s[4][5]. To be brief, the consensus is that there are numerous independent factors influencing developer productivity. Case in point: a recent tertiary literature review by Oliveira *et al.* identifies thirty-five such factors, ranging from soft factors like project management and process maturity to technical factors like development tools and programming languages[6]. This is good news because it implies that many different strategies can be used in concert to improve productivity. On the other hand, this also means that there is an *overwhelming* amount of information available; simply searching on Google Scholar for "software engineering" and "productivity" turns up over 141,000 results, almost all

of which are written by and for computer scientists rather than domain scientists or mathematicians. This implies that it can be time-consuming to operationalize this information, and CSE practitioners must be strategic in their approach. By strategic, we mean that there has to a commitment to routines and norms that bring software science into the loop. In the following section, we give one example of how this can be done in practice.

Rapid Reviews in Evidence-Based Practice

At the Center for Computing Research at Sandia, the recently-formed Software Engineering and Research Department is leading the strategic goal of advancing software engineering research and practice in the CSE domain. It is a hybrid R&D department that combines practitioning with fundamental and applied research in software science, with the aim of being an incubator and an accelerator for impactful ideas. Case in point: we are actively exploring the use of rapid reviews, a technique popular in evidence-based medicine and translated to software engineering through the pioneering work of researchers Cartaxo, Pinto, and Soares[7].

A **rapid review** protocol is a systematic, time-boxed literature review designed to deliver evidence in a timely and accessible way. Rapid reviews are motivated by practical problems and report results directly to practitioners in the field. Equally important is that rapid reviews simplify or omit certain steps from full systematic reviews, enabling turnaround times measured in days rather than months; specifically, rapid review protocols limit the literature search, reduce or eliminate the screening and quality appraisal steps, and present highlights from the literature without formal synthesis. These qualities make it easier to fit literature review into project timelines (e.g. within an agile sprint) and position us to offer research-as-a-service to our customers. To illustrate, we'll give two brief vignettes of department-internal projects where rapid reviews have been applied.

Requirements Elicitation: Our department partners with numerous scientific software teams for embedded development and support. A critical step in that collaboration is to effectively and accurately gather project requirements. This can be difficult for several reasons, such as having to communicate across scientific

domains and having to work with geographically distributed teams. It was unclear what elicitation techniques should be used to work around these challenges. This led to our research question: what requirements techniques have evidence for their effectiveness, and when and where should they be applied, particularly in domain-specific and/or online/remote contexts? Over the course of a week, a single researcher put together a literature review on this subject, including a secondary study of 140 primary studies on elicitation techniques, a survey of 64 requirements analysts on the use of multimedia communication in elicitation, and multiple proposed models for selecting techniques. This information was compiled into an evidence briefing, a short guidebook on effective elicitation techniques and advice on when and how to apply them. The briefing has been used to plan requirements elicitation exercises with our customers.

Software Quality Standardization: The Center for Computing Research hosts a wide variety of software projects ranging from modest research scripts to gargantuan community projects. To support these projects, our department is developing quality assessment and assurance models. From the outset, it was clear that there would be no one-size-fits-all solution, and two questions emerged: whether different teams work better under different sets of quality standards, and what the best way is to "right-size" a software quality model. One researcher performed a rapid review around these questions over a period of two days, including a ten-month observational study of a large telecommunications company, a comparative study of quality factors based on a survey of 175 US companies' chief information officers, and a systematic review of 53 papers on tailoring software processes. The key findings were distilled into an evidence briefing on development factors that are known to influence software quality and productivity. This guide directly informed our next steps, such as the development of a survey tool to evaluate projects in terms of these factors.

In each case, our team faced a practical problem in our work, derived an answerable question from that problem, and compiled evidence from peer-reviewed literature to inform our decision-making. These efforts have been part of an ongoing pilot program, and we are still in the process of evaluating the impact of these and other rapid reviews. However, developer feedback has been very positive. To quote one developer, "I can't tell you how many times I've been in a situation where I wished I had more confidence in the decisions that I made. There's a lot of guesswork and uncertainty. And literature review like this is very valuable"; to quote another, "Showing me how I could apply the information was immensely useful. [...] It saves me time. It saves me from having to bash my head against the wall not knowing what I don't know".

A potential concern regarding rapid reviews is validity. That is, a week-long rapid review cycle will turn up fewer results than, say, a six-month full systematic review, and reducing screening, appraisal, and formal synthesis may also reduce a reviewer's ability to speak authoritatively about their findings. To address this concern, first, rapid reviews are not meant to be a replacement for traditional reviews, but instead play a complementary role; rapid reviews aid operational decision-making in projects where the alternative is usually to do no literature review at all. Second, evidence-based practice is not solely about the research, but instead combines research with practitioners' experiences and values; as in medicine, the assumption is that research can strengthen decision-making and help teams achieve a better state of practice.

Discussion

If the CSE community wants to improve upon developer productivity, software science offers a deep well of insight. However, we need to find ways to put that insight to work in our projects, such as through evidence-based practice techniques like rapid reviews. This, we believe, is not unrealistic. Domain scientists and mathematicians are already trained in the art of searching, interpreting, and applying research literature. Moreover, teams can engage with software engineering practitioners and researchers at their own institutions and/or through academic partnerships. This may also lead to other serendipitous benefits: bringing software scientists into the conversation may spur innovations that address the specific needs and concerns of our community.

As we mentioned previously, developer productivity is a well-established area of research, and this presents actionable opportunities. Teams can select tools, techniques, and processes which are supported by evidence in the literature, put them to the test within their own project, and determine whether that intevention has a measurable impact on productivity. For instance, if a team struggles with creating quality software,

studies may suggest certain techniques that correlate with software quality (e.g. code reviews, static analysis, or test-driven development); that team can then implement and assess those techniques against a relevant measure of quality. Ideally, teams should apply evidence-based techniques like rapid reviews not only to solve new problems but also to periodically revisit past decisions to see if better solutions are now available.

Supporting computational science through software science can shorten innovation cycles and overcome barriers to productivity. Drawing upon established research reduces risk and increases confidence. This is not to say that research is a silver bullet. Research is imperfect and personal judgment and instinct will always play a role in decision-making, but should still strive to treat software with care and rigor. This means seeing software development not just as a tool for science but as a science onto itself — critically evaluating the choices we make about our software on the basis of the best available evidence.

Conclusion

Software is now central to the scientific enterprise. Increasing developer productivity is key for taking full advantage of our computational tools. In this paper, we argue that the question of productivity is a research question, and that software engineering research (or software science) has much to offer in this area. One way of integrating software science into scientific software projects is through a commitment to evidence-based practice, which includes techniques such as rapid reviews. We provided a sketch of ongoing work at Sandia National Laboratories in this direction in the hope of inspiring further discussion.

Acknowledgements

This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. 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. SAND2020-6644 C

Works Cited

[1] Holdren, John P and Shaun Donovan. "National Strategic Computing Initiative Strategic Plan." Technical report, National Strategic Computing Initiative Executive Council, 2016.

[2] Heroux, Michael. "Research Software Science: A New Approach to Understanding and Improving How We Develop and Use Software for Research." 2019 Collegeville Workshop on Sustainable Scientific Software, 2019.

[3] Dyba, Tore, Barbara A. Kitchenham, and Magne Jorgensen. "Evidence-based software engineering for practitioners." IEEE software 22.1 (2005): 58-65.

[4] Bauer, Friedrich Ludwig, et al. "Nato software engineering conference." Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany. Edited by P. Naur and B. Randell. 1968.

[5] Dijkstra, Edsger W. "The humble programmer." Communications of the ACM 15.10 (1972): 859-866.

[6] Oliveira, Edson, et al. "Influence Factors in Software Productivity: a Tertiary Literature Review." *International Journal of Software Engineering and Knowledge Engineering* 28.11n12 (2018): 1795-1810.

[7] Cartaxo, Bruno, Gustavo Pinto, and Sergio Soares. "The role of rapid reviews in supporting decision-making in software engineering practice." Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018. 2018.