Research Software Science: A Scientific Approach to Understanding and Improving How We Develop and Use Software for Research

Development and use of software are fundamental to numerous areas of scientific research. Many scientists write, modify, and use software to gain insight and prove scientific results. At the same time, formal software engineering techniques and knowledge that are widely adopted in other software development domains are not as commonly used in research software projects. In my experience, research software development approaches are more informal, particularly in the upstream activities of requirements, analysis, and design.

One challenge to investing in improved software practices and processes for science is the perception by some that a focus on improving software skills falls under the category of engineering, not science. There is a perception that software engineering—refining repeated practices for more efficiency—is not something that a scientific research funding portfolio should support as a fundamental element.

From this perspective, the best that research software teams can do is spend modest amounts of time learning practices from the mainstream software engineering literature and use them unchanged or with modest adaptation in their own software development. In my experience, however, this approach results in only moderate success—and sometimes even failure. Best practices distilled from larger, more mainstream software domains may be ill-suited for research software teams.

In this article, I propose that the scientific method—which is central to scientific efforts using research software—can be used to study and improve the development and use of research software. Looking at the development and use of software for research through a scientific lens enables us to apply a scientific approach to understand and improve software as a tool for research. In other words, research software development and use are the subjects of our scientific study.

Science applied to research software

As stated above, the primary objective of research software science as I am proposing it is to apply the scientific method to understanding the development and use of research software. This pursuit has strong technical, social, and cognitive components:

Technical component:

The purpose of research software is modeling and simulation of scientific theories; the gathering, analysis, and understanding of scientific data; and related pursuits. These activities typically require years of education in the domain and ongoing community engagement in order to contribute to and keep abreast of new discoveries and approaches. For example, to develop and use computational fluid dynamics (CFD) research software, one must complete years of study in mathematics, physics, and engineering and then continue study of CFD and related fields even when developing and using software.

Social component:

Development and use of scientific software are typically a team effort, increasingly involving more people and more diverse roles. Team interactions, workflows, and tools play a large part in the effectiveness of a research software team. While many people have realized the importance of the technical component of research software development and use, fewer people have focused on the social elements, and even fewer have applied a scientific approach to studying and improving research software team interactions.

Cognitive component:

Improving our approaches to developing and using research software typically requires learning. In my experience, scientists tend to enjoy solving problems. Framing change as a problem or puzzle to be solved can be effective in engaging scientists. Posing improvement goals in a descriptive way, more than a prescriptive, enables scientists to be part of the creative process. More generally, leveraging knowledge from cognitive sciences improves our ability to understand how developers and users of research software approach their work and interact with each other.

Social and cognitive sciences focus

Applying the scientific method to research software teams necessarily involves the social and cognitive sciences. Observations, interviews, data mining, and similar techniques provide the raw materials for analyzing and gaining understanding of important correlations—and ultimately, one hopes, identifying cause and effect—between behaviors, situations, and outcomes.

While the software engineering literature addresses the social and cognitive elements of software development and use, research software teams take on considerable risks by adopting published team practices without scrutiny or adaptation. In my experience, many published team practices are not sufficiently informed by the dynamics and

requirements of research software development and use. To better understand when and how existing software team methodologies are appropriate and to develop new team approaches for research software, we need the skills and tools of social and cognitive sciences applied to research software teams.

Why scientist and not engineer

As mentioned, the software engineering literature provides ample material dedicated to the social and cognitive elements of software teams. Some of this literature is scientific in nature, but much of it is anecdotal, based on years of experience from seasoned software professionals. These writers produce generalized recommendations from specific experiences, often with benefit to the readers.

Software engineering is a large field with many practitioners. However, as Fred Brooks has stated, “An engineer learns in order to build.” An engineering approach does not provide adequate time for transforming the results of engineering best practices studies into scientific outcomes. Surely there are good software engineering researchers who have published many studies; but then this is science, not engineering, and we should call it science.

Why now: Multidisciplinary direction of science

Many important efforts in science require strong multidisciplinary teams. We see that research software is increasingly multiscale and multiphysics, involving modeling, simulation, and data analysis. Adding a scientific approach to understanding the development and use of research software establishes one more dimension in the pursuit of better scientific approaches and is especially appropriate given the growing diversity of scientific teams and the need to understand and optimize team interactions and output.

XYZ software science

One could argue that the scientific method could be applied to any software development and use community. This is true. In my own community, we often focus on high-performance computing (HPC) software as a special subclass of research software. Putting particular adjectives in front of software science, for example HPC software science, helps narrow the focus of study, perhaps leading to more specific outcomes.

Summary

Development and use of research software are rich and dynamic pursuits, worthy of scientific study in their own right. Viewing the efforts as scientific problems opens the

door to applying our skills as scientists to assist in making our software development and use even more effective.

Research software communities of all kinds expect rigorous use of the scientific method as we produce and gain scientific insight from the development and use of software in our domains of study; fluid dynamics research, for example, has benefited tremendously from computation.

We should expect the same rigor as we try to understand how to better develop and use the software tools that enable our research. Research software science—applying science to the study of how we develop and use research software—will qualitatively improve our software capabilities. Forming and promoting the role of a research software scientist should lead to the same qualitative improvement in outcomes we have seen in any other domain where the scientific method has taken hold. Because software is such an important element of research, we expect that focusing on research software science will dramatically improve science overall.

Acknowledgments

The original idea behind this blog post was first presented as a white paper at the 2019 Collegeville Workshop on Sustainable Scientific Software (CW3S19) before being revised and extended for this blog post, which is

crossposted on the BSSw and URSSI sites.

Author bio

Michael A. Heroux is a Senior Scientist at Sandia National Laboratories, Director of Software Technology for the US Exascale Computing Project and Scientist in Residence at St. John's University, MN.