# Dynamics of Scientific Software Teams

Addi Malviya Thakur*, Gregory R. Watson$
*malviyaa@ornl.gov, $watsongr@ornl.gov
*Software Engineering Group, $Application Engineering Group
Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA.

***Abstract*—Scientific Software Teams (SST) are the primary enablers of the transformative science taking place all over the world today. They lead the core development of libraries and packages that enable today's supercomputers to process the vast amounts of data generated by the world's largest and most complex scientific instruments for research and development. This work throws some light on SSTs types and composition, ensuing culture dynamics, common practices, and the evolving skillsets needed to perform next-generation science inventions and discoveries. In addition, the work discusses potential opportunities and threats surrounding SSTs with a focus on long-term sustainable research and development. Finally, this work also discusses best practises, common pitfalls, and practical use cases towards building and managing SSTs.**

***Index Terms*—Scientific Software Teams, Research Software Engineering, Software Excellence**

## I. Introduction

In many scientific domains, the software now underpins the scientific results that are produced. As scientific experiments involving large experimental devices becomes more automated and sophisticated, the software has become the enabler for research, invention, and discoveries. Software is now an essential component in every aspect of scientific research workflows - from data collection, processing, and generation of insights from experimental instruments, to running complex simulation and modeling applications on supercomputers related to climate and nuclear reactor modeling. Software that enables science in this way can be deemed Scientific Software; it allows rapid discoveries, reduces errors and blunders, and ensures reproducibility of complex experiments. The design and development of Scientific Software requires collaborative efforts with diverse skills in software engineering and a deep understanding of the scientific domain. In recent years, the development of Scientific Software has evolved from an individual researcher's effort into a team model. These Scientific Software Teams (SSTs) work side-by-side and often include domain scientists from various disciplines that are integral to the success of the teams' goals and ultimately the scientific results.

In this paper, we discuss various types of software teams from the engagement perspective and their role in designing and developing scientific software and libraries. Software teams continuously evolve and adjust to the changing dynamics of technology and science. Thus a critical aspect of such a team is the agility and flexibility in addressing and incorporating such changes in their workflow. The paper addresses some of these dynamics through the cultural lens and narrates the core principles it is governed by. SSTs are no longer just about creating scientific software; they are also involved in operationalizing them to support production-grade benchmarks such as several thousand users, Service Level Agreements of 99% or more, and mitigating security threats. The SST members also strive to gain substantial domain expertise besides developing a robust software ecosystem.

SSTs inherit several traditional and well-known software development workflows towards developing scientific software. For example, SSTs generally follow agile methodologies that include Scrum, Extreme Programming, Kanban, and others developed in industry, however, these approaches are usually tuned to address the specific needs of scientific software development.

SSTs' interaction with the domain experts lays the foundation for the successful creation of scientific software. Also, the growth of scientific software depends upon how effective interaction and coordination exists among all the team resources regularly.

## II. Related Work

Schmitz et al. discussed a hands-on approach to the overall team dynamics for the PCRaster environmental modeling platform development in [1]. The authors described a successful team consisting of software engineers and environmental scientists working side by side on developing the simulation application. Besides adding new features, the teams were also responsible for maintaining the legacy code. The team grew under the umbrella of the academic environment. While developing the software, the team members were also responsible for writing scientific publications and research proposals and teaching on a need-to-do basis. Several works were recently published that take the "teams of teams" approach to scale productivity and innovation when working on an inter-laboratory program (e.g., exascale) to teach the efficiency and innovation from small teams to aggregate teams of teams[2], [3], [4]. A thorough best practices for scientific computing are discussed in [5].

## III. Types of Scientific Software Teams

There are many possible ways to categorize SSTs, but anecdotally there appear to be two primary types. The first type of team focuses on developing an ecosystem of tools, machines, or data processing pipelines. Such teams' efforts and deliverables are not dictated by a single customer or stakeholder. Instead, their work focuses on applications that cut across multiple projects or even a whole facility. Such teams focus on developing software with a broader user base and their usage as libraries or packages. For example, the design and development of an open-source framework for high performance computing (HPC) network APIs and beyond can be used in many applications and programs to build high-performing and highly scalable network stacks for next-generation applications and systems[6]. In the second type, a software ecosystem is developed that is geared towards solving a specific research problem, and the developers work with the relevant domain experts very closely. For example, Easterbrook et al. and their team developed scientific software that focuses on a detailed case study of climate change-related software development practices[7]. Thus, beyond HPC, a variety of software solves complex scientific problems and usher in an era of discoveries. Such dynamics also make scientific software development different from commercial software development. Scientific software often requires the involvement of a domain expert who is actively involved in the development process.

These two teams can be roughly categorized as the former developing software for more general use cases, while the latter develops software for domain-specific use cases. However, there are some characteristics that apply across both types, such as the type of software development life cycle (SDLC) employed. These SDLCs are often adapted for the type of invention and transformative process that scientific discovery entails, particularly since understanding and defining the problem tends to happen incrementally as the work progresses. For example, in some cases requirement specifications are impossible to collect at the beginning of the work. The requirements emerge as the software is being developed, and the science improves side-by-side. This can mean that approaches such as test-driven development can be difficult to implement end-to-end.

Like any development project, there are often risks other than the knowledge and expertise required to develop the software, such as budget, funding source, and changing priorities. Also, the availability of equipment, data collection, etc., plays a bigger role in the continuation and timely completion of the software development.

Both teams begin the software development process by envisioning solving a specific scientific problem. Issues such as the scaling of applications to support thousands of users, for example, is typically not considered until much later in the process.

The software is itself developed in an iterative fashion and experimental style, since it is often not clear at the outset what will work and what will not. As development progresses, both the domain science and the software may be adjusted in order to achieve intended goal of the research.

Both approaches suffer from the availability of interfaces to do the programming. For example, at times, the scientific application built to run on the HPC hardware is restricted in using C/C++ or FORTRAN programming languages, and thus miss out on the advantages or the availability of high-level programming constructs offered by Java/Python for rapid prototyping and development.

## IV. Interaction and Dynamics

Interaction and group dynamics seem to play a more significant role in the efficient development of the scientific software ecosystem. This is because there is often a greater emphasis among the SSTs to interact and discuss the development process as discoveries are made through experimentation and analysis. In reality, scientific discoveries are rarely planned and are usually the result of chance events, so the development process to create software that underpins the science must also be adaptable to unforeseen events.

Understanding changes and accommodating them is a more significant requirement for software engineers involved in developing the software for scientific research. The Team of Teams model[8] has been shown to work well for large projects that involve coordination and communication within and across organizations. Inspired by the social network concepts, Teams of Teams allow for agility, rapidly adapting to changing dynamics, and information sharing at improved speeds. Like social networks, increasingly frequent and complex communication among the developers evolve to become small managed teams who communicate and share information faster, while still keeping the significant decision-makers informed about progress. Such approaches allow for greater transparency among the stakeholders as well as accountability in achieving shared goals. However, considerable attention needs to be given to scaling the efficiency and innovation delivered by small teams to the aggregate Team of Teams. For example, funding that is shared across multiple teams must translate to effort and participation that benefits the Team of Teams as a whole. Building trust among the individuals and across the teams is essential in order to encourage the free flow of ideas and transparent communication. Other factors, such as the ratio of senior staff (who require little supervision) to junior staff (who require supervision and/or mentoring) must also be considered in order to find the right balance between productivity and managing expectations.

Psychological diversity could accompany technical diversity SSTs culture. Agile practices are starting to be seen as beneficial by the SST community, and are being more widely adopted by SSTs. Recently, several national laboratories have implemented Sprint-style project executions for their Laboratory Directed Research and Development (LDRD) programs to achieve faster results and provide better metrics for assessing the investment made to achieve high profile discoveries. In contrast, the commercial software industry has long known the benefits of using these approaches, and are continually evolving better ways of developing software. As discussed earlier, the nature of scientific software development is rarely predictable, which calls for greater flexibility and higher levels of uncertainty. The agile approach inherently addresses such challenges automatically through iterative improvement and incrementally increasing the level of complexity. The planning that involves a clear discussion of high-level goals for research isolates the implementation and variability succinctly through the agile process, allowing for more remarkable changes and updates along the way as far as the central core research goals are aligned.

Ethics in research is a shared responsibility. While developing scientific software, it is vital to pay attention to ethical responsibility as well. This extends to ensuring that the software that is developed is inclusive, and does not incorporate biases or other forms of discrimination. SSTs also need to consider the implications of their work and the broader societal impact. It is also important that the SSTs operate ethically and responsibly. This may be as simple as ensuring proper attribution for contributors and ensuring that publication decisions have been adequately discussed and verified before releasing any software.

## V. Composition and Skills

SST members vary in their background, knowledge, skills, and abilities. This diversity can be beneficial when it comes to developing software applications for a wide variety of science domains. In the commercial industry, software engineers spend most of their time working on only one project at a particular time. In contrast, in the scientific software community where funding is primarily driven through grants and awards, developers are often working across multiple projects. The fluidity of working on multiple projects, as well for short periods of time, is exhibited across various levels. As a consequence, SST members have only a short time available to ramp up skills and deliver results. This sometimes also puts constraints on the application and implementation of software processes, while still allowing for rapid prototyping and testing of novel scientific ideas, where the focus is on discovery and creation of new knowledge. Scientific software development is also often done for unique problems that have very specific applications, in which case ground-up development has to be done by the members of the team, and external libraries of packages are not always an option. A multi-disciplinary team can be critical to solving the problems in these situations.

The team composition of SSTs should include members with a diverse skill-set as well as experience. The ratio of senior, mid-level, and junior staff needs to be carefully considered in order to optimize development productivity. Senior staff with decades of experience working across multiple domains are needed to be able to drive the science component of the project, comprehend the requirements, and translate them in a format that can be implemented programmatically. As in most software development projects, the senior staff tends to spend more time on the design and planning aspect than the coding and details of the implementation. Mid-level staff who have a good grasp on the architecture of the software play a pivotal role in developing workflows, high-level constructs, technical design. They are involved in creating libraries and critical components of the software that serve as the foundation of the entire ecosystem. Junior staff, including early career, typically require more supervision and mentoring and are devoted to learning and writing code for the software systems. In addition, they learn concepts and workflows from the more senior members of the team and eventually take on more complex and challenging tasks as their skills improve.

## VI. Conclusion

In this work, we addressed several aspects of Scientific Software Teams (SST) and their interplay among the projects, stakeholders, and themselves. Creating effective SSTs is a complex task that requires the active participation of all team members. The nature of the projects, funding, and resource availability play a pivotal role in how SSTs operate, deliver, and perform. Beyond that, core principles such as agile methods, Team of Teams, transparency are the values that bring uniformity and coherency, and dependability to the whole process. SSTs are uniquely positioned to enable transformative research and are critical drivers of inventive science. Thus addressing some of the challenges will not only help nurture this community, but will positively impact every other scientific discipline that uses SSTs in its research and development portfolio.

## VII. Acknowledgement

This manuscript has been authored by UT-Battelle, LLC under Contract No. DEAC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

## References

[1] O. Schmitz, K. de Jong, and D. Karssenberg, “Sustainable scientific software: experiences of the pcraster research and development team.” in *Geophysical Research Abstracts*, vol. 21, 2019.

[2] E. M. Raybourn, J. D. Moulton, and A. Hungerford, “Scaling productivity and innovation on the path to exascale with a “team of teams” approach,” in *HCI in Business, Government and Organizations. Information Systems and Analytics*, F. F.-H. Nah and K. Siau, Eds. Cham: Springer International Publishing, 2019, pp. 408–421.

[3] D. Moulton, E. M. Raybourn, L. McInnes, and M. A. Heroux, “Enhancing productivity and innovation in ecp with a team of teams approach.” Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2018.

[4] B. H. Sims, “Enabling coordinated, distributed development of scientific software: A research agenda for adapting a team of teams approach,” Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2019.

[5] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. Chue Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, and P. Wilson, “Best practices for scientific computing,” *PLOS Biology*, vol. 12, no. 1, pp. 1–7, 01 2014. [Online]. Available: https://doi.org/10.1371/journal.pbio.1001745

[6] P. Shamis, M. G. Venkata, M. G. Lopez, M. B. Baker, O. Hernandez, Y. Itigin, M. Dubman, G. Shainer, R. L. Graham, L. Liss *et al.*, “Ucx: an open source framework for hpc network apis and beyond,” in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. IEEE, 2015, pp. 40–43.

[7] S. M. Easterbrook and T. C. Johns, “Engineering the software for understanding climate change,” *Compu ting in science & engineering*, vol. 11, no. 6, pp. 65–74, 2009.

[8] S. A. McChrystal, D. Silverman, T. Collins, and C. Fussell, *Team of teams: New rules of engagement for a complex world.* Portfolio Penguin, 2015.