The Science of Scientific-Software Development and Use

Workshop sponsored by the U.S. Department of Energy (DOE)
Office of Advanced Scientific Computing Research (ASCR)

Organizing Committee:
Michael A. Heroux, Sandia National Laboratories
David Bernholdt, Oak Ridge National Laboratory
Lois Curfman McInnes, Argonne National Laboratory
John Cary, University of Colorado

DOE Point of Contact: Hal Finkel, DOE/ASCR

Workshop website: https://www.orau.gov/SSSDU2021
Pre-Workshop Report
December 9, 2021

Introduction

Software development and use play a central and growing role in scientific discovery. In some communities, the primary tools are spreadsheets and similar mainstream software adopted for scientific research. In other communities, the specialized hardware, software, and algorithmic requirements call for community-developed software capabilities that are typically designed and written by scientists, who often use the software for their own research, while also providing it to others.

In recent years, the impact of scientific software has increased, with computational and data-enabled science and engineering pervading nearly all aspects of discovery, as complements to experimental and theoretical research. At the same time, scientific software has become more complex due to advances in both computer architectures and next-generation science challenges. As a result, software teams have grown more diverse, including people with computer science skills to assure appropriate algorithm and data structure choices, as well as software engineers to improve software tools, practices, and processes. A variety of community reports have expressed the importance of increasing direct investment in scientific software itself, not just as a byproduct of other research [1, 2, 3, 4, 5, 6, 7, 8]. Consequently, we now have the unique opportunity to fundamentally change how scientific software is designed, developed, and sustained, while addressing urgent challenges in workforce training and recruitment [9, 10, 11, 12, 13].

The art and craft of scientific software development and use have advanced over many decades, primarily benefiting from adapting and adopting advances from outside the scientific community. Advances from the broader software community in programming languages, computer system design and implementation, and software development tools and processes have strongly influenced (in both positive and negative ways) scientific software activities. Research focused on improving software development and use is certainly progressing. Numerous studies continually emerge from the software engineering literature, where authors study practices, methodologies, tools, and human factors with the goal of understanding and improving the impact of software for its intended use, at reduced cost and faster delivery schedule. Even so, the scientific software community is a small part of the overall software ecosystem and, from our observations, has received only modest attention from the professional software community in comparison with other target domains.

Presently, the scientific software community is exploring possibilities to include cognitive and social science, as well as advancement in artificial intelligence (AI), to further improve the effectiveness and efficiency of software in the pursuit of science. The 2021 Workshop on the Science of Scientific-Software Development and Use [14], sponsored by the U.S. Department of Energy (DOE), Office of Advanced Scientific Computing Research (ASCR), focuses on these newer approaches, with the intent to increase and accelerate research on social, cognitive and AI themes as they can be applied to scientific software. Workshop participants and topics were chosen to create an opportunity for identifying and prioritizing research directions toward the goal of expanding the size and diversity of the scientific software community to effectively include these themes.

Scope: Science-based Methodologies for Scientific Software

The software engineering community has a robust R&D component. Conferences like the International Conference on Software Engineering (ICSE) provide numerous examples of this work. Software community leaders such as Steve McConnell provide even further value by synthesizing the literature into usable and impactful advice and practices. At the same time, little direct emphasis is placed on scientific research software relative to other much larger software domains. Our community certainly benefits tremendously from broader community R&D investments, but we believe there are unique opportunities for the *scientific* software community to contribute to and benefit from R&D efforts in the science of research software [15].

Because we are scientists ourselves, our hope is that we can appreciate, incorporate, and expand the role that science can play in improving the development and use of software for scientific research. This hope is at the core of this workshop. Areas of high priority for the scientific community include:

- Advanced design for software products, tools and applications
- Social and cognitive sciences applied to scientific software development and use
- Integration strategies for new ideas and culture change

Advanced design for software products, tools and applications

Various types of scientific software–including reusable libraries, development tools, and scientific applications–all would benefit from improvements in software architecture, flexibility/extensibility, and user experience. The traditional organic nature of scientific software design and development can often lead to impediments for further progress, as the effort required to further adapt the software for exploring the next scientific problem may become large or complicated, or may risk incorrect execution. Furthermore, user needs are often assumed to be known from previous informal experiences and are seldom considered from a formal and objective perspective.

We expect that advances in design strategies will have a disproportionate positive impact on scientific software because the broader software community has seen substantial gains from increased design emphasis, and the scientific software community has only started to consider advanced design approaches.

Social and cognitive sciences applied to scientific software development and use

One of the most promising approaches to understanding and improving software development and use is to incorporate the social and cognitive sciences into our scientific endeavors. The knowledge, tools, processes, and expertise of these communities have impacted many technology development efforts. Books like *Anthro-vision* by Gillian Tett [16] have popularized the important insight that paying attention to human factors is essential if we want to improve individual, team, and community activities in the pursuit of scientific discovery.

As stated above, there is already substantial R&D activity applying social and cognitive sciences to software activities, but very little of it focuses on scientific communities and the peculiar aspects of developing and using software for scientific research relative to other domains. We look forward to good ideas that can be realized in the future.

Integration strategies for new ideas and culture change

Change is hard–for individuals, teams, organizations, and communities. While we have already experienced a great deal of change in software development practices in recent years, there is still substantial room for further improvement. Can we take advantage of the theory of *Diffusion of Innovations* [17], change management processes, and other approaches for insights into how changes take place in our community, as well as guidance for how we might be more effective in facilitating the changes we seek (need) to create?

Similarly, at a more pragmatic level, are there strategies or tooling that would help lower barriers to the adoption of new processes and approaches in software development and stewardship? Are certain processes synergistic and more beneficial when adopted together? Are there ways to package processes together or layer them to facilitate adoption? What characteristics of a project set the stage for easier adoption of new strategies? What kinds of evidence are useful in helping to convince team members of the value of a new process?

Trends

Importance of high-quality scientific software

Scientific software is being employed by a wider community, to inform policy and decision making. At the same time, scientific software is increasing in complexity due to the demands of next-generation science and new computer architectures. This requires larger collaborative teams (and teams of teams), with a broader diversity of skills and perspectives.

Increased use of scientific software by a wider community. Software plays a large and increasing role in scientific discovery across a wide range of domains and computing platforms. These trends are probably most noticeable in relation to computational and data sciences and the application of data-driven methodologies across many scientific domains. In domains where data has always been a central source of insight, important trends are the emergence of large data sources, and computing tools that can take advantage of these sources. Scientists in these communities are increasingly seeking larger computing platforms, including the leadership systems where work has traditionally focused on modeling and simulation.

Data-driven approaches are also emerging as an alternative or complement to modeling and simulation in scientific domains that have traditionally been based on theoretical formulations. These hybrid theory and data approaches enable rich synergies where data-driven inference can learn from and replace slower simulation components in a multiphysics or multiscale computational environment, or can provide sophisticated interpolants trained from high-fidelity parameter space sampling using model-based simulations.

These trends point to the increased importance of scientific software across many scientific domains, implying even greater importance to improving how we develop and use software for scientific research.

Increased use of scientific computing to inform policy and decision making. As computational science and engineering have matured, predictive capabilities are emerging in many domains, moving beyond traditional interpretive simulations. As a result, numerical modeling is being used with increasing frequency to inform consequential decisions and policy making in government and industry. Examples range from ecological and climate-related policies to safety-critical decisions such as nuclear reactor licensing, aircraft certification, the design of bridges and buildings, and epidemiological modeling.

Increased reliance on computationally based results for important decisions justifies, even demands, increased scrutiny of the software and methods on which decisions are based. Further, practical experience suggests that the path from software intended solely as a research tool to its use in consequential settings is frequently both unanticipated and slippery. These concerns are, in many respects, just an extension of conventional concerns about transparency and reproducibility in science, which apply to computationally-based science as well as experimental and observational science. Consequently, we envision a growing need in the computational science community to respond to increasing scrutiny and expectations for software quality, reliability, and credibility.

Increased model and software complexity. The difficulties described above are exacerbated by increasing complexity required to address next-generation computational science. Teams are working toward predictive science and engineering through multiphysics, multiscale simulations and analytics and are addressing requirements for greater scientific reproducibility [18,19]. For example, teams are grappling with the increased model and software complexity required to incorporate higher-fidelity models with more physical processes. It is difficult enough to know how to choose a particular solution component, such as an effective linear solver, for a particular problem. The difficulty becomes much greater in composite systems with multiple components, in trying to understand how to quickly reach a solution for a particular problem in various computing environments.

Increased complexity of computer architectures. Disruptive changes in advanced computer architectures also are causing unprecedented software challenges. The transition to hosted accelerated architectures, specifically nodes with multicore CPUs and multiple GPUs, requires developing new algorithmic approaches, porting code to new compiling and runtime environments, and realizing massive concurrency that is possible only by overcoming large latency bottlenecks. Furthermore, these new CPU/GPU platforms represent only the beginning of the system heterogeneity expected in the future [20].

Increased team sizes and teams of teams. The only way to tackle these increasingly difficult problems will be to bring together multiple teams (or 'teams of teams' [21]) in an efficient and scalable manner. There are, in many cases, natural ways to divide the problem space. For example, we have development teams for solvers, others for build and package management systems, others for visualization, and so forth. The approach of well-defined APIs, either callable or file based, has been successful. But we increasingly see very large projects with a number of components to be developed rapidly, and that includes setting the proper mechanisms of interaction (APIs, for example), which cannot be developed in isolation, as the considerations of multiple subteams need to be taken into account, along with the needs of the user community. Agile methodologies have become popular, but these can be problematic for large scientific computing projects, where at the outset one may not know what to code, as research is needed to determine appropriate algorithms, and then further research may be needed to determine the most performant data structures and methods. An outstanding problem is whether there are methodologies for scientific software development that permit rapid progress in research along with early and incremental delivery.

Increased diversity in needed skills. Software is broadly recognized as a primary means of collaboration across disciplines in computational and data-enabled science and engineering, encapsulating expertise in mathematics, statistics, computer science, and core disciplines of scientific and engineering. Indeed, reusable software libraries and tools have a long history of broad impact, and application-specific community codes are becoming widespread as a means of disciplinary collaboration [22]. Given the continually increasing scope and complexity of collaboration, *software ecosystems* [23] are proving effective, where communities explicitly consider interrelationships among interdependent software products whose development teams have incentives to collaborate to provide aggregate value, where the whole is greater than the sum of its parts. Central to this work are the contributions of *research software engineers* (RSEs) [24], whose expertise in both software engineering and research helps to bridge across disciplines though high-quality software. This broad scope of collaboration also benefits from *project coordinators*, who help software teams to plan work and handle the logistics of coordination. Throughout all of these interactions, experts in social and cognitive science can help us navigate collaborations across aggregate teams, or teams of teams.

History: Progress through newly emerging communities of practice

Addressing these scientific software challenges requires broad community collaboration to change the culture of computational science, increasing the emphasis on high-quality software itself and the people who create it. Responding to these challenges, various grass-roots community groups have arisen in recent years to nurture "communities of practice" [25] in their respective spheres of influence, where like-minded people share information and experiences on effective approaches for creating, sustaining, and collaborating via scientific research software. These groups articulate key issues and needs to stakeholders, agencies, and the broader research community to effect changes in policies, funding, and reward structure, while advancing understanding of the importance of high-quality software in multidisciplinary CSE collaboration and the integrity of computational research [26, 27].

Software Sustainability Institute. An international leader in this topical space is the U.K.'s Software Sustainability Institute (SSI, https://www.software.ac.uk), which has existed for more than a decade for the express purpose of advancing software development and use for scientific research. SSI develops and promotes methodologies for understanding software requirements, building community awareness of the importance of better software, and much more.

NSF SI2, URSSI. The U.S. National Science Foundation has sponsored several programs focused on direct funding for software teams to further develop and support products that have proven broad usability in the scientific computing community, for example Software Infrastructure for Sustained Innovation (SI2) and the more recent Cyberinfrastructure for Sustained Innovation (CSSI). In addition, NSF has sponsored the U.S. Research Software Sustainability Institute (URSSI, https://urssi.us), focused on topics related to the scope of this workshop.

Research Software Engineering Movement. Research Software Engineering (RSE, https://society-rse.org and https://us-rse.org) has emerged as an increasingly recognizable career track, with a growing number of people who consider themselves part of the RSE community. While the definition of RSE varies and other terms have been used to describe this kind of position (e.g., software scientist, scientific programmer), many people in this role increasingly identify themselves as RSEs. Also, the number of workshops and organizations expressly focused on RSE topics is increasing. The growth and institutionalizing of RSE roles as sustainable career paths are in part driven by the goals of improving developer productivity and software sustainability, as RSE skills are essential to achieving these goals.

IDEAS Productivity Projects. In 2014 the U.S. DOE Office of Science, as a partnership between the Offices of Advanced Scientific Computing Research (ASCR) and Biological and Environmental Research (BER), sponsored the creation of the IDEAS Productivity Project. The project expanded in 2017 in the DOE's Exascale Computing Project (ECP, https://www.exascaleproject.org), which requires intensive development of applications and software technologies while anticipating and adapting to continuous advances in computing architectures. Likewise, in 2019 the BER-funded IDEAS-Watersheds project grew out of the original IDEAS project, with emphasis on accelerating watershed science through a community-driven software ecosystem.

IDEAS (https://ideas-productivity.org) [28], as a collection of projects, represents the first direct DOE funding for developer productivity and software sustainability. The funding and support of program managers Thomas Ndousse-Fetter, David Lesmes and Paul Bayer, as well as ECP leaders Paul Messina and Doug Kothe, opened the door to multi-institutional collaboration to foster improving software quality as a key aspect of advancing scientific productivity, while also building communities of people strongly interested in these topics.

IDEAS has benefited from collaboration with SSI, URSSI and similar efforts. The IDEAS initiative has provided the launching pad for various software quality improvement efforts, including the xSDK math libraries (https://xsdk.info) work to advance collaboration and community policies. The xSDK experience has in turn prompted complementary software development kit (SDK) efforts and broader work on the Extreme-scale Scientific Software Stack (E4S, https://e4s.io). The IDEAS initiative spearheaded the Better Scientific Software site (https://bssw.io), a community-based hub for sharing information on practices, techniques and tools to improve developer productivity and software sustainability. Likewise, IDEAS launched the BSSw Fellowship Program (https://bssw.io/fellowship), to provide recognition and funding to leaders and advocates of high-quality scientific software, beginning with DOE support in 2018 and incorporating NSF sponsorship in 2021. IDEAS also sponsors numerous outreach activities (see https://ideas-productivity.org/events), including the webinar series *Best Practices for HPC Software Developers* [29], the panel series *Strategies for Working Remotely* [30], the *Collegeville Workshop Series on Scientific Software* [31], and other tutorials, BOFs, minisymposia, and events.

Emerging transformative technologies

Ubiquitous collaborative software platforms. The rise of software-as-a-service (SaaS) platforms has included a wide range of platforms supporting software development. Although GitHub and GitLab may be among the most recognized such platforms, numerous others offer a wide range of tools related to software development, the management of development projects and teams, communications tools, etc. Many provide a substantial array of services at little or no direct cost.

The widespread availability of such platforms has lowered the barriers to both collaborative and open software development, and through their adoption by many scientific software teams, greatly enhanced the availability and opportunities for contribution from the community. To the extent that projects on these platforms are publicly accessible, they are more easily found by prospective users and contributors. And whether they are public or private, the barrier to collaboration is lowered simply by the reduction in the number of distinct login credentials users need to maintain. But in the context of this workshop, these factors also result in a great deal more (scientific) software being publicly accessible, which facilitates the study of more scientific software, by mining information from the development artifacts thus exposed, in order to better understand the software and the teams who produce it.

Ubiquitous virtual communication platforms. The number of virtual communication platforms continues to explode. For just messaging, it is now common to move from text messaging to email to Slack to the various texting systems provided by social media. Beyond text communication, we broadly use a variety of collaboration platforms, such as Zoom, Microsoft Teams, Google Hangouts, and more. This plethora of mechanisms means that each member of the community must learn the idiosyncrasies of these many tools. At present these tools are generic, with the ability to share screens, view cameras, and have audio communication. Are there ways to make these tools have extensions specific to (scientific) software development?

AI-assisted tools. AI-assisted tools are now prevalent in our content-creation systems. Most of our word-processing environments now include real-time predictive and corrective spelling and grammar support. Similar tools are emerging for programmers such that a programmer can increasingly focus on intent and get support from a pattern-aware AI system to suggest how to specifically implement that intent.

Advances in social and cognitive sciences. In the software engineering community, the field of user experience (UX) is a growing component of software organizations. While UX has always been a part of the software product development cycle, the degree of sophistication has increased, taking into account a growing knowledge base from the social and cognitive sciences. Furthermore, the size and complexity of our scientific software environments continue to grow, making it more important to consider the usability of software within particular environments.

Opportunities

Enabling the continued advancement of scientific discovery

Accelerating scientific discovery. Computation has long been a critical part of scientific discovery, along with theoretical and experimental investigation. Moreover, computation accelerates those methodologies, allowing the design of complex experiments and providing guidance to theorists in parameter regimes not yet reachable. But scientific discovery occurs only when appropriate software exists. This circumstance implies the need for a wide range of computational software that is well designed and well documented, considering the broad context of its use.

Improving developer and user efforts. Developer efforts must first start from domain knowledge. For example, in object-oriented computing, we construct software objects, each of which represents a physical object, such as the electromagnetic field or a charged particle. This very construction requires deep domain knowledge. On the other hand, developers often make use of mathematical libraries, for which they may well understand basic concepts, but efficient use is either not well known or not well explained. A simple example is the use of iterative linear algebra libraries. Small changes in preconditioners can make a huge difference in the number of iterations required for convergence. Yet, there are few guidelines for choosing preconditioners for different types of problems.

It is important for developers to realize that to a software user, the time to solution is the total time, including startup time to learn the software, time to set up the problem within the software, time to debug the setup, time to run, and time to analyze the data and/or visualize it. On the other hand, it does not make sense to create an advanced and sophisticated graphical user interface, which might require a team of multiple developers, to set up a run for some aspect of computational software if that software has only a few users.

Enabling more sophisticated models and advanced use cases. The HPC community continues to move towards greater realism, more accurate computing, the inclusion of more physics, and better coupling between physics modules. For example, an accelerator cavity designer needs to transfer data between electromagnetic modeling software and thermal modeling software. This requires knowledge of different data formats, neither of which the designer controls, and an understanding of the data to be transferred. Work of this sort often ends up being a number of one-offs, which is time consuming. Tools to reduce this work are desirable, with said tools being easy to learn so that the work of learning is more than amortized over the number of transformations to encode.

Expanding the scope of scientific skills and disciplines

Leveraging scientific advances in social and cognitive sciences. Focusing on a scientific approach to understanding and improving scientific software development and use creates a natural impetus to engage social and cognitive scientific experts as part of our software community, leveraging their experience and working with them to understand our software requirements. We anticipate that the integration of these new perspectives, tools, and skills will also benefit the larger software community.

Leveraging advances in AI for advanced tools and methodologies. As software developers and users, we regularly observe, analyze, and execute many similar patterns. The more we can capture these patterns as data sets, the more we can apply machine learning algorithms to expose these patterns for future automated and machine-assisted programming and software use. This approach is already used in many settings but has generally not been extended to important scientific software needs such as Fortran programming, floating-point data, and optimization for performance.

We are already starting to see some of these tools, e.g., GitHub AI programming assistant. We need to understand how these tools will impact scientific software activities. This is both a technical and human factors topic. Based on what we discover, the broader software community can also benefit from what we learn.

Expanding the scientific developer and user communities

Designing higher-level interfaces. Computational science pervades virtually all aspects of our world, including the physical sciences, life sciences, social sciences, and more; many industries already heavily rely on computational technologies to assist with design and manufacturing. However, the effective use of HPC software technologies, even for people whose primary focus is domain-specific science and engineering, tends to require substantial understanding of computer architectures, programming models, and lower-level algorithms and data structures. Enormous potential exists to expand the use of HPC software technologies in industry and decision making. However, a prerequisite is broadening the scope of people who can effectively use these sophisticated tools – requiring the development of higher levels of abstraction and interfaces for non-expert users, while concurrently enabling specialists to customize lower-level choices.

Creating opportunities for new and more diverse scientific community members. Numerous studies have shown that diverse organizations, teams, and communities perform more creatively and effectively—and thus are more productive. While some efforts are already under way to broaden participation in computational science and engineering, our communities could benefit by increasing emphasis on sustainable strategies to advance diversity and inclusion. Work is needed to understand how to improve teaming skills and culture, including leadership of remote, distributed, and hybrid teams, while fully leveraging tools to facilitate distributed work. Equally important is research on building a growth culture in scientific communities, with recognition of the benefits of rich engagement across diverse demographics and areas of expertise.

Understanding how to leverage and adapt knowledge, tools and processes from the broader software development and user communities

Overall, software development and use are fairly well studied by the software engineering research community, and today, perhaps to a lesser extent, by the social and cognitive science research communities. Members of the scientific software community have successfully both adopted and adapted a great deal of knowledge, processes, and tools from the broader software community, but frequently have found ideas that do not translate well, for a wide range of reasons. As a simple example, consider the challenges in applying the Scrum methodology to settings where developers are often neither co-located nor dedicated to the project. But what distinguishes the scientific software development context from more general development is multi-faceted, and itself not well understood.

There are numerous opportunities arising from this situation. First, to understand what distinguishes scientific software development from the more general activity. Second, to better understand (and predict) what knowledge, processes, and tools will or will not translate well into scientific settings. And third, to facilitate the adaptation of general approaches to the scientific software context.

Expanding the impact of scientific approaches

Scientific approaches to learning how to develop scientific software ultimately are of no use without being taken up by the community. It is expected that there will be resistance to changing methods, as there always is. So it is important that from the outset, those studying how to improve methods have thoughts early on about how they would communicate their effectiveness to a wider audience. Naturally, there will be a need for workshops and tutorials, but these are greatly strengthened by showing the effectiveness of new methods in developing non-trivial scientific applications.

Next Steps

Scientific software development and use are typically the means to an end in the pursuit of scientific discovery and advancement. We hope that by emphasizing the potential for impact of scientific approaches to improve scientific software, our community can justify increasing the kinds of skills of scientific software teams (developers, users and other contributors) and the amount of time spent on improving software practices as part of their overall work.

Potential approaches

Approaches that seem attractive include:

- **Partnering with the social and cognitive science communities on methodologies for data gathering and analysis.** The social and cognitive science communities are formally trained in producing scientific understanding by observing and interacting with human systems. While scientific software teams are such systems, and we typically work to make our teams high functioning and improving, our approaches are usually ad hoc and of an engineering nature. If we perceive a need to improve a practice, adopt a new tool, or institute a new workflow, we typically do a quick search for new possibilities, perform an informal evaluation, adopt a new approach, and then move forward. This approach risks missing out on fundamental principles for improvement, and we seldom record why we make particular choices, thereby eliminating the possibility for others to learn from what we have discovered. Partnering with scientific communities who have these skills and experience will help us make better choices and communicate what we learn via publications and other venues.
- **Adapting and adopting knowledge, skills, and tools from the broader software community.** Software engineering R&D investments are substantial in the broader software community. However, some of the assumptions made when gathering requirements or distilling findings to actionable advice are not well suited to scientific software development and use. For example, the members of scientific software teams typically have highly specialized skills, so some assumptions about developer

interchangeability presumed in the broader software community are not feasible for scientific teams. Scientific software teams need to leverage the R&D results from the broader software community but with awareness of what is most promising for our community and how it might need adaptation to be most effective.

- **Iterative and incremental exploration of new ideas.** Another attribute of many scientific software teams is that producing leading-edge science is the team's primary goal. If we are going to be successful in improving the practices of scientific software developers and users, we need to carefully plan how to introduce new practices, processes, and tools. Few scientific software teams can afford to suspend efforts to generate new scientific results, as their funding requires results, and competitiveness with other scientific teams would suffer. Iterative and incremental change enables tuning to achieve a balance of delivering today's results while also improving how tomorrow's results will be produced.

Sample priority research directions

Considering these trends and opportunities, here we introduce a few promising directions of research. A goal of workshop discussions is to consider broad community input to determine priority research directions for the science of research software.

New AI-based tools for improved productivity and sustainability.
AI tools are already improving the quality and reducing the cost and time required to produce effective prose and other written content. Promising research directions for scientific software include exploring which developer tasks are most time consuming and which could be improved by AI tools. While such tools are already under development in the broader software community, creation and integration of these tools for scientific software development and use can be accelerated and adapted to our specific priorities.

Science-based software design methodologies for improved usability and sustainability.
The software development community in general is moving to methods of rapid and incremental delivery. As noted above, there are many unique aspects of scientific software development, including pre-research to understand promising algorithms, data structures, and so forth, given the premium placed on performance in scientific computing. First, what are the usage patterns? Is it the case that scientific computing always follows the steps of computational experiment conception, creating inputs, running, analyzing data, and visualization? A variation on this process is to introduce an optimization loop that wraps the above set of steps with software that selects new simulations. What other patterns are common? Would using AI in such setups improve outcomes? Have such approaches been tried?

Studies should also extend to the users. Looking over a broad range of projects, what are the types of interactions between users and the software? At what point does it make sense for a project to put more effort into usability? Is there a threshold number of users at which such a transition occurs? And is the transition sharp? Or do successful projects ramp up usability over the lifetime of the project? What aspects of usability are addressed first? Is there an endpoint?

Characterizing the attributes of sustainable scientific software. A key concern about software is its *sustainability*. Despite the importance of this concept, it is a rather nebulous term, lacking a clear definition in the context of scientific software. Like in an eye exam, we can often compare and say we think that one software package is more sustainable than another. But in this case, there's no optometrist who can turn those comparisons into a straightforward prescription for sustainable software. Can we develop a working definition of what constitutes sustainable scientific software, without being overly prescriptive? Can we identify the attributes of the software, the development processes, the team dynamics, and other factors that influence the sustainability of a software product? If we want to do a better job of producing sustainable software, it would help to have a clearer idea of the target and how we might get there.

References

[1] M. R. Benioff and E. D. Lazowski, Pitac Co-Chairs, Computational science: Ensuring America's competitiveness: President's Information Technology Advisory Committee. http://vis.cs.brown.edu/docs/pdf/Pitac-2005-CSE.pdf, 2005.

[2] W. Gropp, R. Harrison et al., Future directions for NSF advanced computing infrastructure to support U.S. science and engineering in 2017-2020. National Academies Press, 2016. http://www.nap.edu/catalog/21886/.

[3] M. A. Heroux, G. Allen, et al., Computational Science and Engineering Software Sustainability and Productivity (CSESSP) Challenges Workshop Report, Sept 2016. https://www.nitrd.gov/PUBS/CSESSPWorkshopReport.pdf.

[4] S. Hettrick, Research software sustainability. Report on a Knowledge Exchange Workshop, 2016. http://repository.jisc.ac.uk/6332/1/Research_Software_Sustainability_Report_on_KE_Workshop_Feb_2016_FINAL.pdf

[5] H. Johansen, L. C. McInnes, D. Bernholdt, J. Carver, M. Heroux, R. Hornung, P. Jones, B. Lucas, A. Siegel, and T. Ndousse-Fetter, Software productivity for extreme-scale science, 2014. Report on DOE Workshop, Jan 13-14, 2014. https://science.osti.gov/-/media/ascr/pdf/research/cs/Exascale-Workshop/SoftwareProductivityWorkshopReport2014.pdf.

[6] D. Keyes, V. Taylor, et al., National Science Foundation Advisory Committee on CyberInfrastructure, Task Force on Software for Science and Engineering, final report, 2011. http://www.nsf.gov/cise/aci/taskforces/TaskForceReport_Software.pdf.

[7] J. Dongarra, J. Hittinger, J. Bell, L. Chacon, R. Falgout, M. Heroux, P. Hovland, E. Ng, and S. Wild, Applied Mathematics Research for Exascale Computing, LLNL-TR-651000, March 2014. https://pdfs.semanticscholar.org/08c6/485730d302944967cc6c44762deb2c71c080.pdf?_ga=2.89380702.451571458.1583807553-541362651.1583807553.

[8] J. Hack, K. Riley, R. Gerber, K. Antypas, R. Coffey, E. Dart, T. Straatsma, J. Wells, D. Bard, S. Dosanj, I. Monga, M. Papka, L. Rotman., Crosscut Report: Exascale Requirements Reviews, 2017. https://www.osti.gov/servlets/purl/1417653.

[9] B. Chapman, H. Calandra, S. Crivelli, J. Dongarra, J. Hittinger, S. Lathrop, V. Sarkar, E. Stahlberg, J. Vetter, D. Williams, DOE Advanced Scientific Advisory Committee (ASCAC): Workforce Subcommittee Letter, 2014, https://dx.doi.org/10.2172/1222711.

[10] National Science & Technology Council, National Strategic Computing Initiative Update: Pioneering the Future of Computing, 2019, https://www.nitrd.gov/pubs/National-Strategic-Computing-Initiative-Update-2019.pdf

[11] L. Arafune et al., CI Workforce Development Workshop 2020, https://www.rcac.purdue.edu/ciworkforce2020,

[12] National Science & Technology Council, Pioneering the Future Advanced Computing Ecosystem: A Strategic Plan, 2020, https://www.nitrd.gov/pubs/Future-Advanced-Computing-Ecosystem-Strategic-Plan-Nov-2020.pdf

[13] Office of Advanced Cyberinfrastructure, CISE, NSF, Transforming Science Through Cyberinfrastructure: NSF's Blueprint for a National Cyberinfrastructure Ecosystem for Science and Engineering in the 21st Century: Blueprint for Cyberinfrastructure Learning and Workforce Development, Aug 2021, https://www.nsf.gov/cise/oac/vision/blueprint-2019/CI-LWD.pdf

[14] Workshop on the Science of Scientific-Software Development and Use, sponsored by the U.S. Department of Energy, Office of Advanced Scientific Computing Research, Dec 2021, https://www.orau.gov/SSSDU2021.

[15] M. Heroux, Research Software Science: A scientific approach to understanding and improving how we develop and use software for research, Sept 2019, https://bssw.io/blog_posts/research-software-science-a-scientific-approach-to-understanding-and-improving-how-we-develop-and-use-software-for-research.

[16] G. Tett, *Anthro-vision*, Avid Reader Press/Simon & Schuster, 2021, https://www.simonandschuster.com/books/Anthro-Vision/Gillian-Tett/9781982140960.

[17] E. M. Rogers, Diffusion of Innovations, Simon & Schuster, Free Press, New York, 5th Edition, 2003, https://www.simonandschuster.com/books/Diffusion-of-Innovations-5th-Edition/Everett-M-Rogers/9780743222099.

[18] D. E. Keyes, L. C. McInnes, C. Woodward, W. Gropp, E. Myra, M. Pernice, J. Bell, J. Brown, A. Clo, J. Connors, E. Constantinescu, D. Estep, K. Evans, C. Farhat, A. Hakim, G. Hammond, G. Hansen, J. Hill, T. Isaac, X. Jiao, K. Jordan, D. Kaushik, E. Kaxiras, A. Koniges, K. Lee, A. Lott, Q. Lu, J. Magerlein, R. Maxwell, M. McCourt, M. Mehl, R. Pawlowski, A. P. Randles, D. Reynolds, B. Riviere, U. Rude, T. Scheibe, J. Shadid, B. Sheehan, M. Shephard, A. Siegel, B. Smith, X. Tang, C. Wilson, and B. Wohlmuth, Multiphysics simulations: Challenges and opportunities, International Journal of High Performance Computing Applications, 27 (2013), pp. 4–83. Special issue, https://dx.doi.org/10.1177/1094342012468181.

[19] U. Ruede, K. Willcox, L. McInnes, and H. D. Sterck, Research and education in computational science and engineering, SIAM Review, 60 (2018), pp. 707–754. https://dx.doi.org/10.1137/16M1096840.

[20] J. Vetter et al., Extreme heterogeneity: Productive computational science in the era of extreme heterogeneity, 2018, https://doi.org/10.2172/1473756.

[21] B. Hendrickson et al., ASCR@40: Highlights and Impacts of ASCR's Programs, U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, 2020, https://doi.org/10.2172/1631812.

[22] L. C. McInnes, M. A. Heroux, E. W. Draeger, A. Siegel, S. Coghlan, and K. Antypas, How community software ecosystems can unlock the potential of exascale computing. Nature Computational Science, 1 (2021), pp. 92–94, https://doi.org/10.1038/s43588-021-00033-y.

[23] S. Hettrick, A not-so-brief history of Research Software Engineers, SSI blog, 2016, https://www.software.ac.uk/blog/2016-08-17-not-so-brief-history-research-software-engineers-0.

[24] E. M. Raybourn, J. D. Moulton J.D., and A. Hungerford, Scaling Productivity and Innovation on the Path to Exascale with a "Team of Teams" Approach. In: Nah FH., Siau K. (eds) HCI in Business, Government and Organizations. Information Systems and Analytics. HCII 2019. Lecture Notes in Computer Science, vol 11589. Springer, Cham., 2019, https://doi.org/10.1007/978-3-030-22338-0_33.

[25] E. Wenger, Communities of Practice: Learning, Meaning, and Identity, Cambridge University Press, 1998. ISBN 978-0-521-66363-2.

[26] D. S. Katz, Software in Research: Underappreciated and Underrewarded, 2017 eResearch Australasia conference, Oct 2017, https://doi.org/10.6084/m9.figshare.5518933.

[27] D. S. Katz, L. C. McInnes, D. E. Bernholdt, A. C. Mayers, N. P. Chue Hong, J. Duckles, S. Gesing, M. A. Heroux, S. Hettrick, R. C. Jimenez, M. Pierce, B. Weaver, and N. Wilkins-Diehr, Community organizations: Changing the culture in which research software is developed and sustained, special issue of IEEE Computing in Science and Engineering (CiSE) on Accelerating Scientific Discovery with Reusable Software, 21 (2019), pp. 8–24. https://dx.doi.org/10.1109/MCSE.2018.2883051.

[28] M. A. Heroux, L. C. McInnes, D. E. Bernholdt, A. Dubey, E. Gonsiorowski, O. Marques, J. D. Moulton, B. Norris, E. M. Raybourn, S. Balay, R. Bartlett, L. Childers, T. Gamblin, P. Grubel, R. Gupta, R. Hartman-Baker, J. Hill, S. Hudson, C. Junghans, A. Klinvex R. Milewicz, M. C. Miller, H. Nam, J. O'Neal, K. Riley, B. Sims, J. Shuler, B. Smith, L. Vernon, G. Watson, J. Willenbring, and P. Wolfenbarger, Advancing Scientific Productivity through Better Scientific Software: Developer Productivity and Software Sustainability Report, ECP Technical Report ECP-U-RPT-2020-0001, Jan 2020, https://exascaleproject.org/better-scientific-productivity-through-better-scientific-software-the-ideas-report.

[29] O. Marques, D. E. Bernholdt, Best Practices for HPC Software Developers: The first five years of the webinar series, Oct 2021, https://bssw.io/blog_posts/best-practices-for-hpc-software-developers-the-first-five-years-of-the-webinar-series.

[30] E. Raybourn, R. Milewicz, D. Rogers, E. Gonsiorowski, B. Sims, G. Watson, Working Remotely: The Exascale Computing Project (ECP) Panel Series, July 2020. https://bssw.io/blog_posts/working-remotely-the-exascale-computing-project-ecp-panel-series.

[31] K. Beattie, G. Chourdakis, H. Cohoon, V. Dyadechko, N. Fadel, C. Ferenbaugh, R. Jacob, J. Lofstead, R. Milewicz, D. Moulton, J. Moxley, T. Munson, S. Osborn, W. S. Pereira, S. Shende, B. Smith, J. Willenbring, U. M. Yang, S. Yates, S. Knepper, L. C. McInnes and M. Heroux, Software Team Experiences and Challenges: A Report from Day 2 of the 2021 Collegeville Workshop on Scientific Software, Oct 2021, https://bssw.io/blog_posts/software-team-experiences-and-challenges-a-report-from-day-2-of-the-2021-collegeville-workshop-on-scientific-software.